ABSTRACT

Title of dissertation: LEARNING TASK MODELS
FOR ROBOTIC MANIPULATION
OF NONRIGID OBJECTS

Joshua D. Langsfeld, Doctor of Philosophy, 2017

Dissertation directed by: Professor Satyandra K. Gupta
Department of Mechanical Engineering

As robots become more prevalent in smaller manufacturing and maintenance settings, it will become important to enable them to learn new tasks quickly without explicit programming by a human. One particularly challenging domain in robot learning is handling nonrigid objects and materials such as fluids and easily deformable parts and tools. The complexities of modeling nonrigid systems make it infeasible in general for a robot to plan its actions to perform a task by simulating their behavior, requiring an ability to learn an unknown model through experience. This experience can be gained both from a human demonstrating the way to perform a task and the robot itself performing task attempts to incrementally improve its model. Over time, as more experience is acquired, the robot should eventually obtain a model that allows it to perform the task when faced with new variations, generalizing its past experience.

This dissertation explores this problem in the context of two robot tasks: pouring a specific volume of fluid into a moving container, and cleaning stains off

of compliant objects. First, an approach is presented to learn the parameters of the pouring task by observing human demonstrations. The model learned from the demonstrations can then be exploited to learn how to pour new volumes with minimal extra learning effort by the robot. Second, this same task is used in development of a general approach for autonomous learning. Here, the robot takes a small set of random samples from the parameter space to build an initial task model and selects new parameters to test by building many local linear models. As more data is acquired, the robot's task performance improves substantially and it is able to very quickly find solutions to new task variations. Then another approach is shown that uses demonstrations to estimate a cost function for performing the task. This enables the robot to also learn strategy elements from how humans perform tasks. Finally, two approaches are discussed to learn the deformation model of a compliant part. A bimanual setup with two robot arms is used to hold and clean the part and the model is used to optimize the plans for both arms to reduce cleaning time and deformations. The first approach shows a black-box learning method to directly predict the part deformation when a known force is applied. The second uses a finite-element structure to represent the part, and learns the model by updating the stiffness parameters. When given a new part, the system only needs a few trials to improve quickly enough to clean new stains efficiently by predicting how much the part will deform under cleaning force.

LEARNING TASK MODELS FOR ROBOTIC MANIPULATION OF NONRIGID OBJECTS

by

Joshua D. Langsfeld

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2017

Advisory Committee:
Professor Satyandra K. Gupta, Chair/Advisor
Associate Professor Nikhil Chopra
Assistant Professor Mark D. Fuge
Assistant Professor Rodolphe J. Gentili
Professor James A. Reggia, Dean's Representative

For all who consider me family

Acknowledgments

In reaching this point in my life, I owe enormous gratitude to many individuals and groups; too many to mention or even count. Nevertheless, there are several people that I must single out for special attention.

First, sincere thanks to my advisor Professor Satyandra K. Gupta for his unflagging determination and indispensible guidance over the more than five years I have been able to work with him. It was always a pleasure to watch him plan a clear path forward whenever it became too nebulous to see clearly what was happening. Observing his leadership of both me and a large, unruly research group provided a great model that I'm sure will continue to be strong influence for me many years into the future.

I would also like to thank Professor Jim Reggia for providing me an opportunity for collaboration with his research group and for serving on my doctoral committee. Thanks also to the other members of my committee, Prof. Nikhil Chopra, Prof. Mark Fuge, and Prof. Rodolphe Gentili, for their valuable insights, feedback, and suggestions for this work. Additional thanks to the Office of Naval Research for their generous financial support of the work contained in this document under grant N000141310597.

I would also like to thank the many great people who I have had the pleasure of working with over the past years. Thank you to all my coauthors and labmates, especially Prof. Krishna Kaipa, Ariyan Kabir, Brual Shah, Michael Kuhlman, Iain Brookshaw, Petr Švec, Madan Dabbeeru, Sagar Chowdhury, Lena Johnson, Galen

Mullins, Pradeep Rajendran, Shaurya Shriyam, and Di Zeng. Thanks to my former colleagues at Energetics Technology Center, Bob Kavetsky and Bill Hinckley, for taking a chance on an out-of-the-blue candidate and bringing me to Maryland in the first place. Thank you to all the wonderful people at past institutions who certainly had no small part in shaping the trajectory of my life and career for the better, especially Prof. Dawn Tilbury, Prof. Ella Atkins, Dhananjay Anand, Jeff Fletcher, William Harrison, and Steve Vozar. Thank you to the fantastic support staff at the University of Maryland and others, who worked tirelessly to keep everything behind the curtain working seamlessly. And deepest thanks to all my family and friends, who have always been enormously supportive and eager to see me succeed.

Finally, I must provide special thanks to my wife, Ramya, who has never been anything but a supportive and loving partner. She has had the dubious honor of living with a grad student prone to procrastination for over three years but took everything in stride and with a sharp wit. When looking back on this period, I'm sure what will stand out above all are the good times we always had.

Table of Contents

List of Figures

Chapter 1: Introduction

1.1 Motivation

The use of robots is currently experiencing an explosive phase of growth. New developments in hardware capability and computational processing power have made the idea of having challenging tasks performed by robots realizable. Robots have successfully been programmed to execute many tasks that are far more unstructured and complex than classic industrial tasks where the exact sequence of actions is usually predefined. Some impressive specific examples include a robot folding cloth in a domestic environment [1], a robot arm playing table tennis [2], and autonomous vehicle navigation in cluttered urban environments [3].

However, programming robots to execute these complex tasks can still be quite time consuming and challenging. Generally, manual programming approaches rely on an expert programmer formulating a simple version of the problem and applying a search-based planning algorithm to discover a solution. To generate a problem within the conceptual grasp of a human programmer, simplifying assumptions may be required and might include constraints such as considering all objects being manipulated to be rigid and ignoring dynamics. However, when such assumptions are not feasible, as in the case where the task demands a full analysis of deformable

or fluid materials, the development cost can be immense. For example, the programmer may be forced to perform many iterations of programming and testing before successfully specifying a valid version of the problem for a planner to solve. Additionally, state-of-the-art planners require far more computation power to solve problems involving fluids or deformable bodies due to the complex nature of their dynamics. However, many routine tasks in industry and our daily lives involve such complications [4]. Representative examples include measuring and transferring quantities of fluid, or handling deformable objects such as cable or flexible plastic. Formal representations that enable robots to perform complex tasks with such objects may take far too long for a programmer to specify and may not be solvable in real-time using this traditional approach.

A particular environment where these issues are pertinent is an industrial setting where robot manipulators work on mass production lines. Due to the large overhead in manually programming industrial manipulators, robots are only used for very large production runs where the savings in each individual task performance can sum to and exceed the significant cost of manually programming the robot to do the task [5]. However, small and medium manufacturers (SMMs) whose needs revolve around short production runs with fast-changing requirement tasks, find the cost of programming the robots to be highly prohibitive. Moreover, in some cases, it could take a long time to program them relative to the task duration typical of a short production run. Therefore, methods that are able to substantially reduce the time, cost, and effort required to program a robot for new tasks have the potential to

open up many more opportunities for robots to become widespread, thereby creating significant additional value for society at large.

1.2 Goal and Scope

This dissertation explores some of questions that are currently relevant for development of a robotic learning system with the goal to acquire the ability to perform tasks involving nonrigid objects. The learning system has an overall structure that includes a model of the task that can be improved through experience and attempts of the task. For many classic robot tasks, the task model is straightforward to define. However, especially in the case of nonrigid object manipulation, there are many details that are still areas of active research. In particular, this dissertation explores the research questions of how the task model should be parameterized, how the learner can effectively use its current task model to select new attempts than will quickly lead it to task successes, and what task aspects can be transferred directly from human demonstrations.

These question fall under the overarching conceptual themes of this work, which are illustrated by the diagrams in Figs. 1.1 and 1.2. Figure 1.1 shows a rough conceptual represenation of a full robot learning system. Note that there are two possible sources of new knowledge for the robot to improve its task model: human demonstrations, and experience from task executions. A fully-developed system deployed in the world would most likely integrate both of these approaches into a unified system, taking advantage of the ability to learn jointly from demonstrations

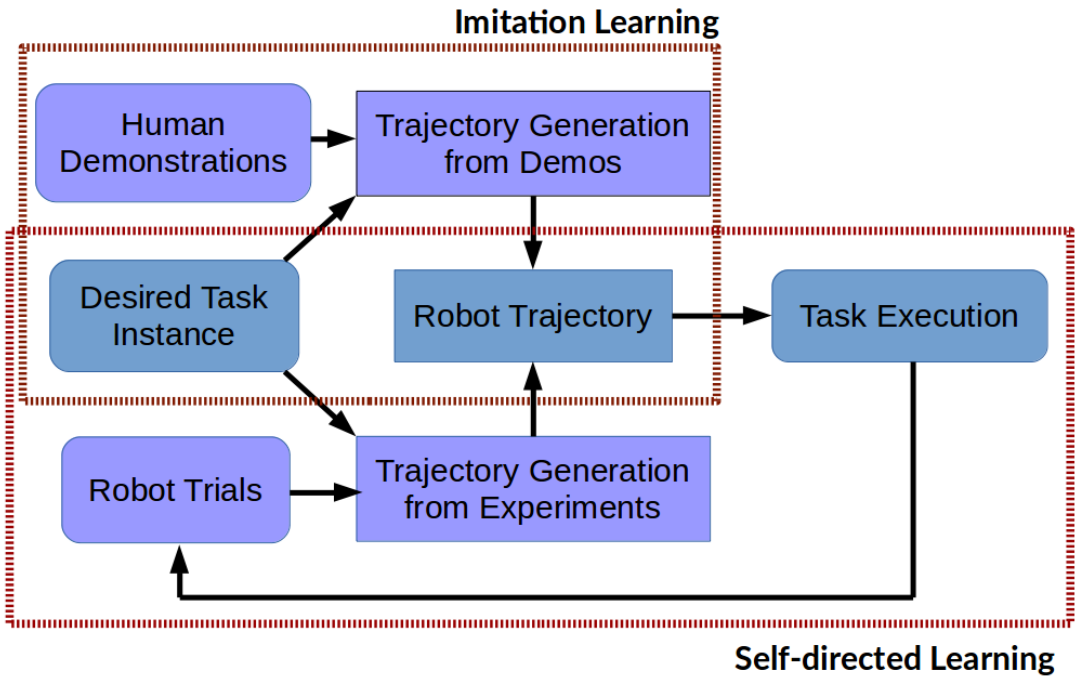

Figure 1.1: Conceptual overview of a robotic system that can learn from human imitation and self-directed experience.

and experience which has been shown to be more effective in the case of humans learning motion tasks [6]. Here, we explore the two paths independently to delve deeper into the details of what each one entails. Additionally, this work makes a distinction between two major methods of generating the robot trajectories for new tasks, as seen in Fig. 1.2. We distinguish between direct trajectory generation by selecting parameters from a dynamics model versus using a more traditional planner which is operating on a model of the object interaction dynamics. The choice of method here mostly depends on the particular task being done by the robot, as certain tasks are more suitable for one or the other.

In this dissertation, several approaches are developed to explore the interaction of these major themes. These approaches are demonstrated on two representative

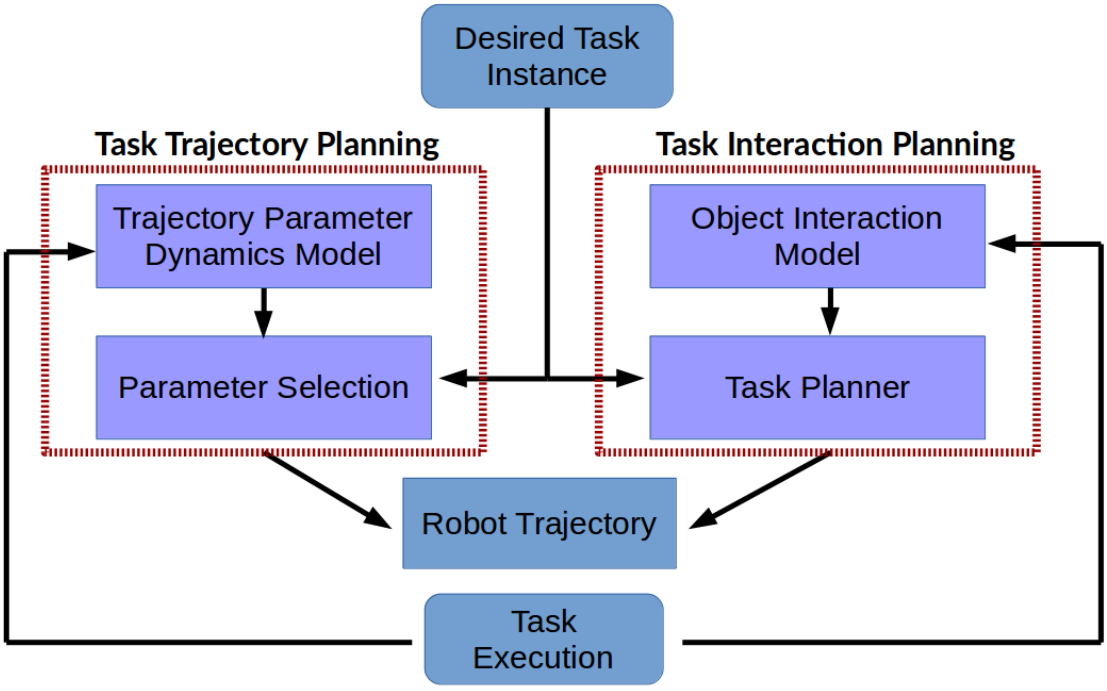

Figure 1.2: Two different methods for generating trajectories for new task instances.

tasks: pouring a specific volume of liquid into a container, and cleaning stains off of compliant objects. The specific focuses and contributions of the dissertation are further detailed as follows:

- *Incorporating Failure-to-Success Transitions in Imitation Learning for a Dynamic Pouring Task*: In the manipulation of fluids, it is not feasible to rely on physics-based models for real-time computation of the task dynamics. This is especially true in the case where a specific volume of fluid must be poured into a moving container within a certain time period, implying a planner cannot make use of any pseudostatic flow rate behavior by moving slowly. However, humans can quickly learn the dynamic fluid behavior of the task through a set of trials, both successful and not. This chapter presents an approach for

building a model of the fluid pouring task from a set of human demonstrations and using the model to identify parameters for a new task variation. It is relevant to the research issues of how to parameterize tasks involving fluids, how to generate a model of the task from human demonstrations, and how to use the model to select trajectory parameters for the robot in order to successfully learn a new task variation.

- *Selection of trajectory parameters for dynamic pouring tasks based on exploitation-driven updates of local metamodels*: Continuing with the dynamic fluid pouring task, this chapter discusses an autonomous learning approach where the robot builds an initial model from a set of random experiments and incrementally improves the model by selecting new points in the parameter space to test. The main focuses of this chapter are selecting a function approximation algorithm for representing the task model and an exploitation strategy for selecting new test points with the goal of rapidly achieving task success. Results on both a synthetic function approximation task and the pouring task show the approach is able to find solutions to new task variations quite quickly and the average number of test points needed drops as more data is acquired, indicating the robot is capable of long-term learning.

- *Task cost function estimation from demonstrations in cleaning of elastically deformable objects*: This chapter switches the focus to bimanual cleaning of elastically deformable objects. Performing cleaning tasks on compliant objects requires the robot system to anticipate the behavior of the part when in contact

with the tool and update the part model as it gains additional experience. An important aspect of this process is acquiring a balance between task execution speed and the amount of stress applied to the part, especially when the full part dynamics are still unknown. The approach presented here uses human demonstrations of cleaning previously unknown parts to estimate the cost function controlling this balance. The main contribution is an algorithm that can estimate cost function parameters dependent on part features or dynamics even when the demonstrator is not fully aware of them.

- *Online Learning of Part Deformation Models for Robotic Cleaning*: This chapter continues exploration of the task of cleaning compliant objects with a bimanual robot setup. One arm grasps a part while the other holds a cleaning tool and moves it across the part surface while applying a normal force. Since the part stiffness is quite low, the grasping location must be optimized to minimize the deflection caused by the cleaning tool force. The optimization is done using a deformation model of the part which is learned through experience as the robot performs initial cleaning attempts. The predicted deformation is modeled using a nonparametric technique, allowing for arbitrarily complex models to be learned over time. Results show that even when the robot is given no prior knowledge of the part deformation behavior, it is able to quickly infer a model with sufficient accuracy to correctly optimize both the number and locations of the grasping positions. This chapter further explores the issue of approximating a task model and additionally shows how

the learning algorithm can take advantage of known structure in the model through the parameterization.

- *Learning a Finite-Element Part Model during Robotic Cleaning of Linear Elastically-Deformable Objects*: The final chapter expands on the parameterization of the previous chapter to use a finite-element model for predicting deformations. The finite-element model allows the system designer to bring in prior knowledge about the system characteristics which can help the learning process. In particular, it greatly speeds up the learning time for part with different stiffnesses in different regions over the black-box approach used previously. The primary contribution of the presented approach is an update algorithm that is used to locally adjust the stiffness parameters of the element mesh whenever new data. Simulated and physical results show the learning system is able to use data from multiple attempts to successfully perform new task variations.

Chapter 2: Literature Review

2.1 Overview

Extensive prior work has been done with the goal of enabling autonomous learning for robots resulting in a vast field of literature with many different branches and aspects. The work discussed in this dissertation makes use of much of this prior knowledge both directly and through inspiration of certain features. This chapter, therefore, attempts to provide a succinct overview of the different subfields of learning for robotics which are relevant to this dissertation, and to provide sufficient context for the work presented here. The different areas covered are:

- Reinforcement learning (RL) - Reinforcement learning is one of the more established fields of learning and there have been many successful applications to robotics. This discussion here focuses on the core abstract ideas of RL and previous applications to robotic manipulators in particular.

- Active learning (AL) - Whereas reinforcement learning typically has the goal of succeeding at a task by learning a model, active learning has the explicit goal of learning the task model itself. A brief overview of the ideas and applications of active learning in robotics is presented.

- Imitation learning (IL) - By observing a human or other agent perform a task, the robot can obtain initial knowledge about a task model faster than starting from scratch. There has been significant prior work done in imitating humans for manipulation tasks.

- Modeling methods - At the core of all learning algorithms, the robot must store the acquired knowledge about the task it is learning and use it to build a model that can predict the task dynamics in new scenarios. This section discusses some of the major modeling methods used during the learning both from the robot's own experience and through imitation.

Additional discussion is also provided on past results achieved on similar tasks and using similar systems to the ones demonstrated in this dissertation. These include:

- Automated trajectory generation for manipulation tasks.

- Manipulation of fluids.

- Automated cleaning of surfaces.

- Manipulation of deformable objects with unknown characteristics.

2.2 Reinforcement Learning

2.2.1 Classical Reinforcement Learning

Fundamentally, reinforcement learning enables a system to learn new tasks solely though feedback from a reward signal. A comprehensive overview of the basic elements of classical RL is given by Sutton and Barto [7]. Typically, a system will perform many task attempts, or *episodes*, and receive a certain positive or negative reward after each attempt. An important distinguishing aspect of reinforcement learning is that the reward is given sporadically, as a single real number after each task attempt, for example, and is not a continuous signal always available to the system. This implies that the system must operate for periods of time without immediate reward feedback and must therefore infer what actions are likely to lead to positive rewards at future times.

The core components of a reinforcement learning system are (1) a state description, (2) a set of available actions for each state, and (3) a policy that determines which action to take in each state, either deterministically or stochastically. Classical reinforcement learning typically assumes that the task can be modeled by a Markov decision process (MDP), which requires that the next state the system arrives at is determined solely by the system's current state and the current action it selects. This assumption is generally reasonable for robotics applications as movement and manipulation tasks usually do not involve complexities such as remembering what actions were taken at the beginning of the task attempt when deciding the current

action. Using a MDP model, a policy could then be formulated by learning a *value function* and selecting the action that maximizes the value function at the system's current state.

2.2.2 Value Function Approximation vs. Policy Search

Since the learning system does not have prior knowledge of either the task dynamics or what actions and states will lead to positive rewards, it must have some internal model that is improved with additional experience. This model is used to generate the current policy. There are two typical approaches for how the internal model is represented, which determines the nature of the possible learning algorithms: value function approximation and policy search.

Classical RL generally uses a value function, which attempts to capture the value of every state represented in the MDP, accounting for the expected rewards that will be accrued after selecting optimal actions starting from that state. Estimating the value function is especially straightforward when the system has a finite and discrete state space, in which case it can be stored as a look-up table and adjusted whenever the system gains further experience at each state. For continuous state spaces, the value function must be approximated with a finite set of parameters, for example, with a radial basis function network [8] or fourier basis [9].

The advantage of using a value function representation is that it can be learned via dynamic programming through Bellman's principle of optimality [10]:

$$V^*(s) = \max_a \left\{ R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \right\} \tag{2.1}$$

which shows that the value function at a state (s) is simply the immediate reward gained from the optimal action (a) summed with the values of the possible subsequent states from that action, weighted by their probabilities. Gamma is a discount factor ($0 < \gamma < 1$) that places more emphasis on current rewards, which is a requirement for the algorithm to converge and not continue to search for rewards infinitely far into the future. Since the value function has this recursive relationship, it is straightforward to implement learning algorithms that can work solely on the state transition model encoded in $P(s'|s,a)$ to converge on the true function such as value iteration and policy iteration. The primary disadvantage of using a value function formulation, however, is that it requires the system to visit the full state space in order to have a fully correct approximation. This is often highly problematic for robotic systems, with high dimensional continuous state spaces.

The alternative learning formulation is policy search, where the system learns directly a set of parameters (θ) that define a policy that maximizes the cumulative reward:

$$J(\pi_\theta) = \sum_{s,a} P^{\pi_\theta}(s,a) R(s,a) \tag{2.2}$$

This formulation often works well for robotic systems as it avoids the primary disadvantage of the value function approach. The complexity is also generally lower, since θ is usually represented by much fewer parameters than the approximation of the value function. However, the tradeoff is that there is no general principle of optimality that can determine the optimal policy parameters. For this reason, policy search algorithms often only have the goal of locally optimizing a policy starting from an initial value of θ, such as gradient following [11]. In practice, however, this limitation does not preclude robots from finding policies that are good enough, as determined by the human superviser. In general, guaranteeing to find the optimal policy for a real-world task is highly intractable and even humans often converge on suboptimal policies when learning new tasks. Therefore, robotics has seen much more usage of policy search algorithms.

2.2.3 Applications for Robotics

A common application of reinforcement learning for robotics is for the control policy for manipulation tasks [12]. Many control policies are based around dynamic motion primitives [13], [14], which encode the motion characteristics of a task into a dynamic system that directs the manipulator at any point in the state space. Some results include learning DMP policies for tasks such as smoothly hitting a drum [15], hitting a ball off of a stand with a bat [16], and hitting a table tennis ball [17] [18]. Besides learning the parameters of DMPs, other systems have been demonstrated effectively, such as autonomous maneuvering of a helicopter [19].

As mentioned previously, these approaches generally rely on policy search variations of RL, commonly with policy gradient algorithms [20] [16], where the parameters of the policy model are adjusted locally. Kakade et al. [21] demonstrate a related approach where the behavior of a continous state-action pair space is assumed to be approximated by local models surrounding a finite set of samples. They prove bounds on the optimality of such a model but the problem of defining the ideal finite covering set remains specific to the application. Deisenroth and Rasmussen [22] show a policy search algorithm that uses a Gaussian Process model to approximate the system dynamics and can be learned with much less data than comparable approaches. They demonstrate their algorithm on the control of low-cost manipulator with significant actuation noise [23].

Mihalkova and Mooney [24] present a hybrid approach where an agent undergoing reinforcement learning can request relocation to a different region of the state space and can actively select the most promising state to reduce its task cost. This allows the system to accelerate its learning substantially.

Finally, Buchli et al. [25] present a policy search algorithm designed to scale to higher dimensions by relying more on the known structure of the manipulator dynamics model. They tailor their approach to doing force control using DMPs and demonstrate the robot learning to perform a movement with minimum effort.

2.3 Active Learning

Active learning (AL) is another related area for this dissertation, with the overall goal of learning a task or policy model by selecting new query points that lead to the fastest improvement of the model quality. In contrast with reinforcement learning, AL does not make use of a reward signal that indicates task success. Instead, active learning approaches have the explicit goal of building an accurate task model and formulate optimization criteria for the learning algorithm using some metric of uncertainty about the true model. The system continues the learning process until the metric drops below a given threshold [26].

Fundamentally, the system is given autonomy to select where in the task space to collect more data to reduce the overall uncertainty. In the standard formulation as described by Settles [27], the learning system has access to a small set of labeled data and a much larger set of unlabeled data known as the *pool*. It is assumed that labels can be easily obtained for any sample in the pool but with a nontrivial cost, constraining the system to obtain as few labels as necessary. The goal of the learner is to select samples from the pool such that the model it generates using the labeled data set is sufficiently accurate for the purposes of the application.

Most approaches involving applying a ranking strategy of the samples in the pool to select the next one that will either improve the model the most. The main strategies are briefly summarized. Query-by-committee strategies [28] make use of a set of models competing for different hypotheses and the next query is made on the sample that causes the most prediction disagreement among the models. Expected

error reduction [29–31], selects the next sample such that the expection of model generalization error is minimized. Variance reduction [32] selects the query that minimizes the variance of the model during prediction. Expected model change [33] selects the sample that is expected to cause the most substantial change in the model. The metric may also defining directly on the labeled data set to simplify the system by bypassing the model. Examples include using a similarity measure with the known data [34] and defining a trust-region around the known sample points [35].

AL approaches have been applied to a variety of problems in engineering and robotics where mathematical models are difficult to obtain. Some examples include nonlinear system identification [36], inferring robot's own morphology [37] and that of other robots in the environment [38], learning the inverse kinematics of highly redundant bionic arms [39], learning environment's degrees of freedom [40], and robot grasping [41].

When faced with the constraint of real-time task learning, learning algorithms generally must make some tradeoff between model exploration and exploitation [42] so as to not spend excessive time learning unusual task dynamics that are highly unlikely to appear during normal task execution. Several learning approaches rely on exploratory movements to find the model structure prior to direct learning [43], [44], even using a different experimental platform than the final learning one [45]. Examples where exploration is used heavily during the learning process include finding a helicopter control policy [46], during reinforcement learning of movement primitives [47], and in developing an initial sensor model [48]. In contrast, Rosales

et al. [49] show a exploitation strategy to learn additional material properties from what information is already available.

2.4 Imitation Learning

Imitation learning (IL) has been a widely explored field of research, especially popular in the domain of robot manipulators [50–52]. The two more general problems that are typically addressed in approaches using IL are (1) defining appropriate behavior of the robot in previously unseen regions of the task space where no demonstration data exists and (2) compensating for the suboptimal demonstration quality typically provided by operators in real-world environments.

For generalizing demonstrations to unseen situations, initial work in robot manipulators focused on extracting sequences of important points from the demonstration trajectory. This is seen in work such as keyframe-based learning from demonstration [53] and contact mapping for whole-body grasping [54]. These approaches typically involve the demonstrator providing the optimal trajectories and are limited in the amount of generalization possible. An alternate strategy that does not involve generalizeing trajectories involves a higher level of extraction such as the particular type of grasp used [55] or the use of the segmented subparts of the manipulator object [56].

More recently, much work has focused on the idea of generating a dynamic system which will provide a desired direction for the manipulator joints or the end effector at every point in the workspace. Typically, the dynamic system will be

defined by a Gaussian mixture model (GMM) [57], or other similar empirical model extracted from the set of demonstrations [58,59]. The work of Khansari and Billard [60] demonstrates a method by which the dynamic system can be learned from multiple demonstrations and constraints imposed such that the resulting system has global asymptotic stability towards the trajectory end point. Similar approaches include the dynamic motion primitive (DMP) [13,61,62] which relies on mixing both linear and nonlinear systems with the linear component responsible for ensuring stability. While the original formulation used only a single demonstration to define the DMP, recent work has extended it to allow many [63]. These dynamical system methods are effective for providing a robust ability to imitate the demonstration at all points in the space. However, they still assume the human is providing optimal demonstrations and suboptimal ones can only be rejected by averaging them with many others [64].

Other approaches to compensate for the scarcity of demonstrations involve learning autonomously after just a few demonstrations. By having the robot explore the task dynamics, less supervision is required from the operator. This approach has been used successfully for tasks such as flipping a pancake [65] using reinforcement learning techniques [66]. In general, the robot engages in a gradient-like search of its policy parameters [67] while repeatedly attempting the the task. The primary disadvantage is the requirement of an explicit reward signal to help the policy search, which may be highly task dependent. This makes it more difficult to deploy these algorithms in a generic setting.

In terms of handling suboptimal demonstration quality, approaches often solve an inverse optimal problem, where the system learns the implicit cost function the demonstrator is operating under. One of the first landmark results was the successful training of autonomous helicopter trajectories by learning from an expert pilot [68]. Here, the demonstration trajectory was assumed to be a noisy measurement of a true, optimal, trajectory and estimation theory was able to extract the ideal trajectory for the helicopter to follow. This method is robust for training a complex dynamic system to follow a specific desired trajectory but is unable to address the first issue of providing a general policy when the state is far from the desired trajectory. This learning mechanism was also successfully demonstrated for the problem of estimating costs of a certain vehicle parking style [69]; a similar noise-rejection method was used for manipulation tasks [70].

Another learning from critique style approach avoids suboptimal demonstrations and achieves generalization by using instructor feedback as seen in [71, 72]. Rather than the human giving full demonstration trajectories, they observe the robot planning and executing its own trajectories and halt the process as necessary, providing corrections of single elements. This enables to robot to learn an approximation of the hidden optimal cost function for the task iteratively. While effective for certain tasks, the robot is unable to learn from a full example trajectory provided by a human and instead must be restricted to local learning.

Finally, there is existing work that directly performs learning from failure, where the approach relies on observing a human perform failed demonstrations only [73]. Here the approach uses the same Gaussian mixture model framework,

and relies on the system to explore regions of the task execution space away from where the demonstrator failed. This was successful for dynamic, single-dimensional tasks but had limited ability to scale into higher dimensions. The authors suggested better performance could be achieved by incorporating additional knowledge of the demonstrator beyond just the failures.

2.5 Model Approximation Methods

This section provides an overview of the different modeling methods that are at the core of learning algorithms. Fundamentally, the model is a function that predicts the behavior of a task in unseen regions of the state space and so function approximation methods are often used to generate the model from a set of collected data points. In the robotics community, the current most-commonly used function approximation algorithms are based on linear models and Gaussian processes. This section presents an overview of the main techniques used for both of theses approaches, as well a few others. More details can be found in the thorough overview of modeling methods for robotics done by Nguyen-Tuong and Peters [74].

2.5.1 Gaussian Process Methods

2.5.1.1 Gaussian Process Regression

GPR is a standard approach for estimating a nonlinear function from few data samples and has been used successfully in many motion learning tasks, both with approaches learning the task value function [75] and dynamics function [22]. GPR

fits a Gaussian process over a set of training samples comprised of multidimensional inputs and real-valued outputs, and produces a predicted normal distribution for the output of any point in the input space. While a Gaussian process can be thought of as an infinite-dimensional normal distribution, when making predictions for new input points the formulation depends only on the training data, which makes the algorithm simple to implement. The standard training and prediction algorithm is derived by Rasmussen and Williams [76] and gives efficient prediction for specified points in the input space.

The standard algorithm takes a data set of previous trials, with inputs $\mathcal{X} \subset \mathbb{R}^n$, and outputs $\mathcal{Y} \subset \mathbb{R}$, and returns a predicted mean (μ_*) and variance (σ_*) of the output for a test point, $x^{(*)}$. The most important component of the GP is the covariance function, $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, which controls the similarity of neighboring points in the input space and enforces the smoothness of functions that are fit to the training data. The most common selection is the standard squared exponential covariance function of the form

$$k(\mathbf{x}, \mathbf{y}) = \sigma_f e^{(\mathbf{x}-\mathbf{y})^T D(\mathbf{x}-\mathbf{y})}. \tag{2.3}$$

Here D is a symmetric, positive-definite matrix that defines a distance metric. Typically a diagonal matrix is used with each diagonal element as a free parameter that controls how quickly the influence of two points drops as the distance between them increases along a particular axis. The other free parameter used is σ_f, which is a uniform scaling factor for the covariance. Additional details on the exact com-

putations used for training the model and for making predictions are provided in Appendix A.

The main advantages of GPR are its nonparametric nature and simple analytical formulation, allowing for accurate modeling of highly complex functions provided sufficient training data. Additionally, a Gaussian process inherently is able to account for noisy training data, compensating for significant training outliers through the influence of other nearby points and the mean function. The main disadvantage is the significant computation cost required for training, which makes it harder to use in real-time for large amounts of data. Additionally, the model requires a mean function which acts as a prior and so in regions of the parameter space without training data, the predicted function values will always return to the mean function.

2.5.1.2 GPR Variants

As mentioned previously, the primary cost in using standard GPR is inverting the covariance matrix created from the training samples. Matrix inversion is a $O(N^3)$ operation and becomes computationally intractable with data set sizes that are easily acquired with current hardware. Several approaches have appeared to address this issue, which mostly fall into two areas: local GPR and sparse GPR.

Local variants of GPR aim to reduce the computational complexity by having multiple models that are only influenced by nearby data, as in LWR. As more data is acquired, more models can be generated, which keeps the computational cost of regression for each individual model low. An important observation is that, whereas

the local variants of GPR are more computationally efficient, their performance in terms of learning and prediction accuracy is generally inferior to that of the original GPR.

For example, Nguyen-Tuong and Peters [77] proposed a local GPR variant (LGPR) with a training complexity of $o(n^3/M)$, where M is the number of local models. The local behavior is triggered by having a new model created whenever incoming data is at a sufficiently high distance from the existing models in the feature space. In all three test cases using highdimensional data from real robot experiments, they reported results that showed that the approximation performance of LGPR was better than LWPR, but inferior in comparison to GPR. Similarly, Meier et al. [78] presented a local variant of GPR that combined the improved computational performance with similar accuracy levels as GPR. On the popular SARCOS inverse dynamics learning task, they showed that both local and global versions of GPR do slightly better than LWPR in terms of accuracy and in most cases, the global GPR is still better than local GPR. They used the incremental sparse spectrum GPR [79] that uses a typical GPR optimization procedure for offline training as their global GPR method. The authors reported that the performance of global GPR degraded as the offline training data became less representative of the online test scenarios. Data sets based on ryhthmic motions of a KUKA robot at various speeds was used in these experiments.

Alternatively, sparse variants of GPR [80] maintain a global approximation behavior but learn on a representative subset of all the data, enabling the algorithm to scale to higher data sizes. Grollman and Jenkins [81] used sparse Gaussian

Processes to model the control policy for a dog robot to perform soccer tasks. The core of the algorithm involves checking each new data point as it arrives for its novelty contribution and if above a certain threshold, using it in a small set of basis vectors that define the model. If the novelty is below the threshold, the algorithm does a slight update of the existing basis vectors instead. However, as expected, the prediction accuracy of sparse GPR is not as high as standard GPR. Furthermore, the authors obtained real-time learning performance by imposing a limit of the size of the basis vector set, which may prevent the system from fully learning complex policies over time.

In summary, local and sparse variants of GPR mainly improve upon the speed but not prediction accuracy of the global GPR on representative datasets. The core tradeoff involves the number of data points that must be used in learning and how fast the model must be trainable. With modern computing hardware, the tipping point seems to be (very roughly) around 1000-2000 data samples with any sort of real-time learning rate. When dealing with fewer samples, standard GPR provides the most power but other selections must be made when the data sizes go above this threshold.

2.5.2 Linear Model Methods

2.5.2.1 Locally Weighted Regression

Locally weighted regression (LWR) is a set approaches that share the common feature of using multiple linear models with local influence [82]. Typically, the

models behave in the same manner as a single linear least squares regression models, but the prediction of each model is weighted by some distance measure to the test point, giving a general prediction structure of

$$f(\mathbf{x}) = \mathbf{x}^T \beta \tag{2.4}$$

where β is the vector of parameters that are learned in model training and is found by minimizing the training error:

$$C(\mathbf{x}) = \sum_{i=1}^{N} L(\mathbf{x}_i^T \beta, y_i) K(d(\mathbf{x}_i, \mathbf{x})) \tag{2.5}$$

where, $(\mathbf{x}_i, y_i)$ are the training data, K is the weighting function, d is the distance metric, and L is the loss function. The variation in different LWR approaches can then come from changing the L, d, or K. Some standard selections for the distance metric may be the Euclidean distance or simply a unique value associated with each data point. The weighting function is typically a smooth kernel function with either a fixed or variable bandwith parameter, or it might simply select the k-nearest neighbors and give zero weight to any other points.

LWR has been used in several robot control [83] and manipulation applications, including control of series-elastic actuators [84], gravity compensation when holding objects [85], and executing periodic motions [86].

2.5.2.2 Locally Weighted Projection Regression

LWPR is a variant of standard LWR, designed for efficient incremental learning with high dimensional data [87], and popular for robotics applications [88]. LWPR provides a method of incrementally learning nonlinear functions through the use of linear models that have local influence only. LWPR is nonparametric and generates new local models as needed throughout the data space to cover the full domain of the function to be learned. Each linear model, however, does not have the full dimensionality of the data but rather, uses partial least squares to detect which directions in the input space are relevant to the function behavior. This allows the model to learn fewer parameters than a more brute-force approach, enabling efficient incremental learning ability. Furthermore, the locations, distance metrics, and number of linear models are all adjusted by the learning algorithm as more data is available.

The local form and parameters of the model can be seen in the primary function prediction equation, which is a weighted average of the local models:

$$f(\mathbf{x}) = \frac{\sum_{k=1}^{K} w_k(\mathbf{x})\Psi(\mathbf{x})}{\sum_{k=1}^{K} w_k(\mathbf{x})} \tag{2.6}$$

The weights on the models are calculated using a Gaussian kernel function parameterized by a center position, $\mathbf{c}_k$, and distance metric, D_k which controls the activation value of test points.

$$w_k(\mathbf{x}) = exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{c}_k)^T D_k(\mathbf{x}-\mathbf{c}_k)\right) \tag{2.7}$$

The models themselves are linear, projected into a lower-dimensional space as determined by the behavior of the target function in the vicinity of the model center. The linear model and projection equations are:

$$\Psi_k(\mathbf{x}) = \beta_0 + \sum_{i=1}^{R} \beta_i s_i \tag{2.8}$$

$$\begin{aligned} s_i &= \mathbf{u}_i^T \mathbf{x}_{i-1} \\ \mathbf{x}_i &= \mathbf{x}_{i-1} - s_i \mathbf{p}_i \\ \mathbf{x}_0 &= \mathbf{x} - \bar{\mathbf{x}} \end{aligned} \tag{2.9}$$

The parameters R, β_0, β_i, u_i, and p_i are computed statistically and on-the-fly from incoming data. More specific details on the implementation and methodology are available in [87].

2.6 Trajectory Generation for Robot Manipulation

There are a diversity of different approaches to specifying robot trajectories with complex model dynamics in the literature. We focus on illustrating past work involving robot manipulation tasks using model and trajectory parameterizations and approaches to generate trajectories that correctly perform specific tasks.

Many researchers have used trajectory optimization to find solutions for robot manipulation tasks, typically optimizing a given geometric path with or without a model of the task dynamics [89], [90]. Mordatch and Todorov demonstrated the use of trajectory optimization to help train a neural network-based manipulation control policy [91] for 3d reaching movements. However, in general using a standard trajectory optimizer is quite expensive if the task dynamics are complex. This remains true with parallelizable optimization methods such as genetic algorithms [92]. Kim et al. [93] developed a trajectory generation method which is able to learn a model of the task dynamics and compensate for unobservable elements in simple object manipulation tasks. Their approach formulates the trajectory creation as a policy in the state space which is approximated by multiple local trajectory generators. A similar approach involves sequencing the solutions of low-dimensional optimization sub-problems [94].

Posa and Tedrake address the problem of optimizing the planner parameters of a geometric path for robustness in manipulation tasks involving frictional contact [95] where the trajectory parameters are the contact reaction forces. Luo and Hauser provide an extension to transform the parameters of the trajectory into a fast linear

programming optimization problem [96]. This allows the robot to perform tasks such as sliding a stack of blocks across a rough surface without any falling down.

Zhang et al. [97] used a sampling-based motion planner to find trajectories for a ball throwing task [98] by searching for valid intermediate dynamic states. They were able to iteratively adapted the planned trajectory to match expected behavior with relatively few attempts. Learning is also possible by storing full paths in a trajectory library [99] and adjusting the most appropriate path to work when given a new task instance [100], or by having a planner compensate for different instances having learned the relevant invariant constraints [101].

2.7 Manipulation of Fluids

An initial result in pouring is from Akgun et al. [53] where a robot learns to scoop and pour quantities of coffee beans directly from human demonstrations. The robot can learn the task elements from a small number of demonstrations but does not have the ability to generalize to changes in the environment. Pastor, et al., [102] showed an initial result of learning a pouring task using DMPs. Their system shows generalization in the pouring target location where the robot proceeds to empty its held container into the target container without spilling.

Later results used reinforcement learning techniques to pour either an entire bottle of fluid [103] or a target volume [104] while simultanously learning to avoid spilling. Again, DMPs are used to encode the robot motions and the learning is done on the DMP parameters. The core contribution is using a statistical analysis

to reduce the dimensionality of the learning space, thereby greatly speeding up the learning rate. In simulation of a stationary pouring task, Nemec et al. [104] achieve no spillage after an average of 7-8 attempts, with a pouring error around 10 milliliters.

Rozo et al. [105] demonstrate an force-based approach that enables the robot to pour volumes of fluid with different initial volumes inside the bottle being held using Gaussian Mixture Regression. The pouring behavior is encoded with a parametric hidden Markov model. They successfully showed the robot could learn to pour a desired volume of liquid into multiple target containers starting from a new initial volume with just a single prior demonstration.

The work of Brandi, et al. [106] shows the generalization of a pouring task to new objects (fluid source and target) not seen in the initial training phase given by kinesthetic demonstrations. The robot can determine a correspondence between the new object and the training object by finding the warping between the object geometries. This warping can then be applied to the manipulator motion to successfully pour out a full quantity of fluid into a new object. Other possible generalization includes learning the behavior of different pouring containers [107].

Other work focuses ocused on combining different pouring skills to generalize to new scenarios instead of the acquisition of the skills, which are created manually [108] or from demonstrations [109]. They investigate the differences in pouring behavior depending on the type of poured material, the container shape, the initial container pose, and the target pouring amount. Their approach relies heavily on planning

using knowledge from the existing skill library and measurements from a new task context, which enables a large amount in generalization to new task variations.

In summary, researchers have successfully been able to implement learning algorithms for specific pouring tasks but there are still open questions in terms of the algorithm abilities to generalize to new task instances. There does not appear to be existing work that addresses the problem of learning a pouring policy for a task instance that allows for precisely pouring different volumes of fluid.

2.8 Robotic Cleaning

The main themes relevant for the work in this dissertation are autonomous cleaning with a bimanual robot system and using force and position sensors to do online learning of object compliance properties. Both areas have had significant prior results obtained.

Multiple robotic cleaning systems have been previously developed, primarily to clean floor surfaces using a mobile platform [110], [111], or to clean surfaces in constrained environments with few degrees of freedom available for robot movement [112]. For the more general case, robots that attempt to clean an area in a short time typically rely on a coverage planner to generate motion paths, with certain regions of the surface(s) required to be uniformly passed-over with a cleaning tool [113]. In the work of Hess et. al. [114], the identified stained regions are then reevaluated after each cleaning pass to produce a probabilistic state-transition model to represent the cleaning performance. We take inspiration from this approach in continually

updating our task model as the robot gains cleaning experience. However, our work is different as we use separate models for the different objects to be cleaned, which are nonparametric. A different example is the work of Do et. al. [115] that shows an approach to learn a nonparametric model of cleaning performance for a wiping task with a humanoid robot. However, their approach involves learning the trajectory by the robot arm and is therefore unsuitable for integration with a coverage planner. King et al. [116] used equilibrium point control to generate wiping behaviors for a robot giving bed baths to patients. The robot performed the wiping task using relatively low forces (< 3 N) without consideration of deformation model of human skin.

Finally previous work not covered in this dissertation but with minor contributions from the author include studies on learning parameters for the cleaning tool model [117] and on planning algorithms for fully covering stained areas on parts with complex geometries [118]. In relation to the work presented here, however, other work has been focused on cleaning rigid objects only. As far as can be determined by the author, no previous work has addressed the problem of cleaning surfaces of objects with unknown deformation characteristics.

2.9 Manipulation of Deformable Objects

Many research results have been obtained using approaches in manipulation of deformable objects, both elastic and inelastic [119] [120]. Initial work is primarily focused on planning approaches for manipulating objects such as cloth or string

into desired goal states [121], assuming that the planner has correct knowledge of the object deformation behavior. Several approaches have been used specifically for objects with linear geometry, including topological planning [122], minimal-energy curvature planning [123], and motion planning incorporating clamping and aligning reference points on the object [124].

2.9.1 Learning deformation model

For elastic or near-elastic materials, task models require more predictive ability than simply the resulting geometry of the object and typically provide some method of predicting deformation in response to external forces. A straightforward method is to use a black-box machine learning algorithm to directly map inputs to outputs, done previously to predict the response of a soft sorting table [125], or of simulated biological tissue [126]. Other approaches have developed models from physics principles, including networks of masses and springs [127] [128] [129], and a mass-tensor model for three-dimensional deformations [130]. Finite element methods to predict deformation [131] are also feasible and can be highly accurate, but are less common due to high computational cost for large models [132], though this can be mitigated through offline precomputation in certain cases [133] [134].

There have been multiple prior approaches developing methods for online learning of these deformation models for newly seen objects. Specific previous results focus on estimation of the stiffness parameters from visual data of example deformations [135], direct touch from the robot [136] [137] [138], or both [139]. Mul-

tiple results also been seen for learning the model properties of tissue during surgical procedures [140] [141]. These approaches share the characteristic of learning the parameters of an assumed structure in the model, or of a prespecified feature vector measurement through a perception system [142]. The work of Berenson [143] shows a somewhat different approach where the robot controls induced deformations without a model by assuming a property of *diminishing-rigidity* where the local behavior of points on the objects are determined by their distance to the robot gripper.

2.9.2 Imitation Learning Methods

Imitation learning techniques can also be used to learn the dynamics of a deformable object manipulation task. This can be done at a high geometric level [144] where demonstrations on objects such as rope [145] and cloth [146] can be mapped onto new task variations with the objects to assist the trajectory planner. Alternately, the system can record the force interactions, possibly through a haptic teleoperation interface [147], and used to determine models for interaction dynamics of the task [148] or the controller used to drive the system towards a goal or stable state [149]. Another use of demonstrations is to extract strategy elements of the task that are transferable to the robot [150], [151]. Often, the strategy can be transfered as a cost function or metric [152] that quantitatively influences the behavior of the task planner.

Finally, this work is also related to previous work done in learning task models from demonstrations involving force interactions. Results include learning models

for flipping objects in a cooking pan [65], and engraving patterns in a plastically deformable substrate with a rigid tool [153]. These and other similar approaches typically use a nonparametric modeling technique to learn arbitrarily complex behavior. However, the model learning is generally oriented towards finding an optimal set of policy parameters that minimize a task dependent cost function, whereas we only require that the model is accurate enough that excessive deformation does not occur and impose no penalty on model inaccuracy otherwise.

2.10 Summary

This chapter discusses some of the extensive prior work done in learning for robotics. Many impressive results have been obtained but there are still substantial open research questions. Two major issues are representation and generalizability. It is often desirable to represent a task to the robot with some prior knowledge or else it becomes impossible to learn in any reasonable amount of time. However, too much prior structure can be significant factor in constraining what it is possible for the robot to learn and so it should be determined how the ideal initial structure might be acquired automatically. Additional, there are many open questions on the best way to enable the robot to generalize its experience to novel settings where it has no explicit training data. Many of existing methods are unable to fully generalize to all variations of a single task, setting aside even the possibility of using acquired knowledge for related tasks as humans can naturally do without issue.

Chapter 3: Incorporating Failure-to-Success Transitions in Imitation Learning for a Dynamic Pouring Task

This chapter is derived from work presented at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS): Workshop on Compliant Manipulation, 2014 [154]

3.1 Introduction

Robots are rapidly becoming more prevalent in industrial settings to perform repetitive tasks more reliably and efficiently than humans. However, current robots still require substantial time and cost investment in programming the robots to perform new tasks, strongly limiting which firms are able to make use of them. A promising alternative to the traditional manual programming approach is imitation learning [50–52], which is becoming increasingly popular over the recent decades as a new method to program robots. The underlying idea of this approach is to allow an agent to acquire the necessary details of how to perform a task by observing another agent (who already has the relevant knowledge) perform the same task. Usually, the learning agent is a robot and the teaching agent is a human. Often, the goal of imitation learning approaches is to extract some high-level information

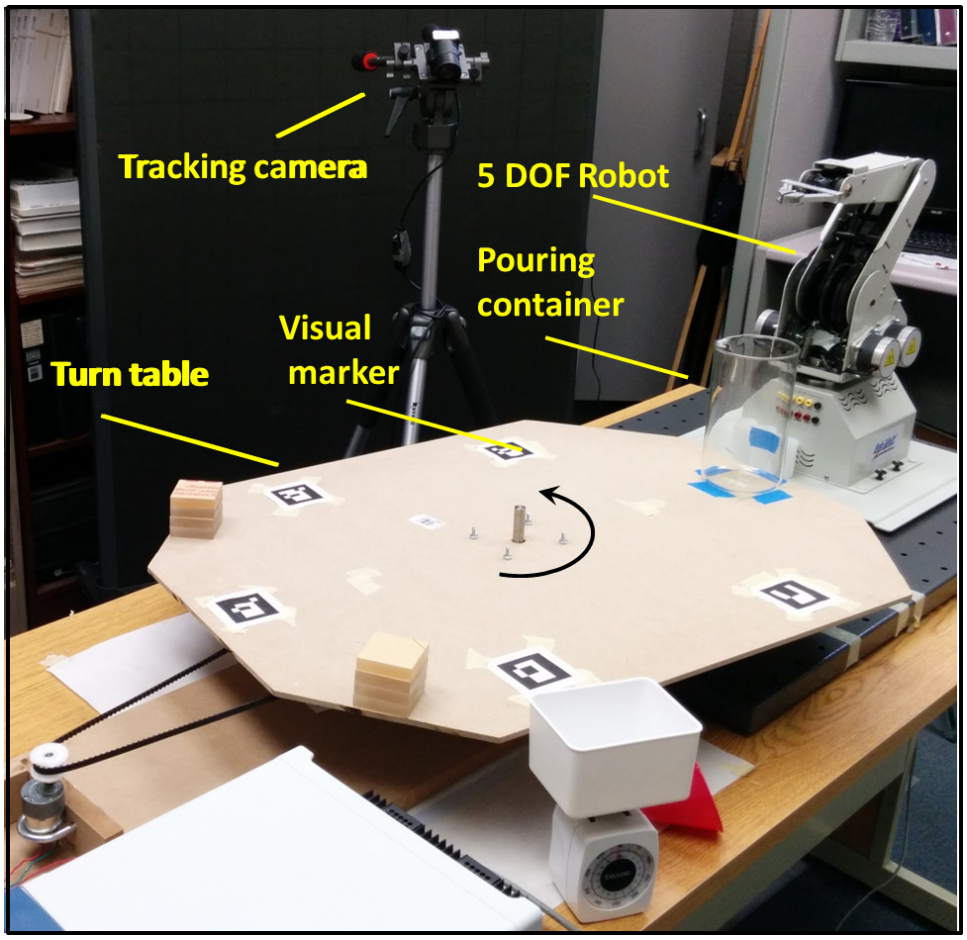

Figure 3.1: Physical setup used to carry out the pouring task experiments.

about how to perform the task from a sparse set of recorded demonstrations, which is independent of the particular variables that change for each task performance.

However, most traditional approaches to imitation learning in the robotics area only utilize successful human demonstrations (e.g., work in [155] assumes human provides optimal demonstrations). These demonstrations are used to construct a model that identifies parameters to be used by the robot in doing the same task. If the robot is unable to do the task using the parameters prescribed by the model, then the approach fails. The reasons for failures are often insufficient number of demonstrations or the subtle differences between the robot and the human that are not modeled. This phenomenon is generally referred to as the "correspondence problem" [156,157] in the imitation learning and cognitive science communities. Unfortunately, relying on a sufficiently large number of demonstrations or a model that captures all the differences between the human and the robot is simply impractical. A robust approach to imitation learning is needed that anticipates failures in the transfer of skills from the human to the robot and has built-in features to recover from it.

Human operators often need to perform challenging tasks multiple times in order to be able to perform them at the acceptable level of performance. Typically, under motor challenge, human performance is highly contaminated by errors during early learning stage. In-turn, throughout multiple trials humans use this motor error to adapt their neural command in order to learn the proper motor coordination. This learning process can be quite fast, but the human may continue to have errors and failures in his or her trials [158,159]. As mentioned earlier, previous

approaches to imitation learning in robotics area have mainly relied on successful demonstrations. This chapter focuses on different approach to imitation learning. In addition to learning from successful demonstrations, we are also interested in learning from errors made by human operators and how they recovered from these errors in subsequent trials.

In the approach described in this chapter, we learn simple rules from human demonstration that capture how human demonstrators change parameters to transition from failed demonstrations to successful demonstrations. If the robot fails to do the task using the prescribed parameters from the transition boundary, it changes parameters using the learned rules and tries again. This capability enables it to keep trying until it succeeds.

In this chapter, we present our imitation learning approach for a fluid pouring task. This is a difficult task for the traditional programming method due to the importance of the current fluid dynamic state. A planner that could define a proper path for the arm would need to integrate with a full computation fluid dynamics (CFD) simulator in order to evaluate feasible path candidates [160]. The planner would therefore be prohibitively expensive to use in a real-time planning system and would entail significant development effort even if it could be made to run sufficiently fast. Therefore an imitation learning approach is used to achieve the goal.

The task considered in this chapter involves grasping a bottle containing a fluid and pouring a specified amount of the fluid into a container placed on a rotating table. The experimental setup is shown in Fig. 3.1. The intention behind coming up with this task is to simulate an automated production environment where the

robot would be required to perform a similar task repeatedly and as fast as possible. A human first demonstrates how to successfully perform the pouring task. For a successful demonstration, the human must correctly determine how much, and how fast, to tilt the bottle in order to begin the pour. Additionally, the human must constantly track the moving container while pouring, and determine when to stop before the container exits the task workspace. These decisions will depend on (1) the table rotation speed, (2) the amount of fluid that must be poured, and (3) the initial amount of fluid in the pouring container. These variations of the task can make it challenging even for a human and preclude the possibility of simply recording the operator's behavior at one setting and using it for all others. Instead, the learning algorithm must extrapolate the correct set of parameters for use in a novel situation based on the parameters used by the human in other similar situations. This allows the robot to perform the task under large variations without a complicated planning or perception system. Achieving this using the manual programming approach would be more costly with many iterations of programming and testing. Experimental results show that imitation learning is a good approach to enabling the robot to solve this task.

3.2 Overview of Approach

We take the following approach to imitation learning. In addition to learning from successful demonstrations, we are also interested in learning from errors made by human operators and how they recovered from these errors in subsequent trials.

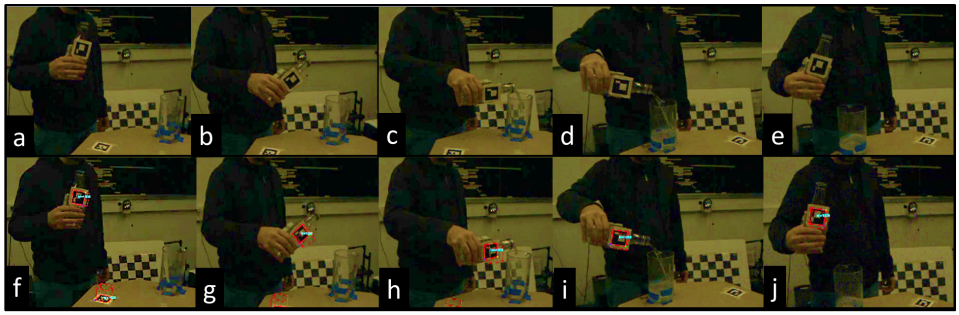

Figure 3.2: (a-e) Snapshots from a video footage of the human demonstration of the pouring task. (f-j) Snapshots from the corresponding video showing the visual tags detected by the tracking system.

Every human trial in our approach is classified as either a successful or unsuccessful demonstration. Every unsuccessful demonstration is scored using a penalty score (e.g., amount of water poured beyond the target). We define a finite-dimensional parameter space to capture the essential characteristics of all possible demonstrations. We can compute the transition boundary between successful and unsuccessful demonstrations using a support vector machine (SVM) classifier. This boundary represents non-dominated successful demonstrations. Theoretically, a point on this boundary prescribes parameters to be used by the robot to successfully carry out the task.

However, in practice using a point on the transition boundary does not always mean success for the robot because of the following two reasons. First, the transition boundary is constructed using a limited number of human demonstrations, and the parameters defining the space may not fully characterize the demonstrations' performances. So the constructed boundary is an approximation of the actual boundary. Second, there are differences in the morphology of the robots and the human demonstrators. So there might be subtle differences between the behaviors

of robots and humans as they try to execute a task with the same set of parameters. So when a robot tries to execute a task based on the prescribed parameters of the transition boundary, it may not completely succeed (e.g., it is close to being successful, but the execution leads to a non-zero penalty score).

In our approach, we learn rules the demonstrations that capture how the humans change parameters to transition from failed to successful trials. If the robot fails to do the task using the prescribed parameters from the transition boundary, it changes parameters using the learned rules and tries again. This capability enables it to keep trying until it succeeds. In summary, our approach gives the robot an informed set of initial parameters to try and carry out the task. It also gives the robot rules that describe how to change the parameters if the initial set of suggested parameters does not work. Therefore, the main contribution of the work is an initial investigation into how learning can be done to simultaneously take advantage of failures and successes. Two algorithms are proposed to leverage this knowledge.

3.3 Pouring Task

This approach is illustrated by performing imitation learning on the pouring task. The pouring task configuration is defined by the following three parameters:

1. Target pour amount p. We assume tolerance of $\pm\ \epsilon$ around this nominal value.
2. Moving container speed v
3. Amount of fluid in the pouring container f

The goal is to complete this task in the small time window when the container is reachable without spilling the liquid. The task is successfully completed if

1. The amount poured in the container is between $p + \epsilon$ and $p - \epsilon$.

2. No fluid is spilled.

If the task cannot be successfully completed, then we assign a penalty score. The penalty score is the amount of fluid that is outside of the tolerance range (i.e., negative number in case of under pouring and positive number in the case of over pouring).

Based on our initial exploration, the following four parameters need to be selected to carry out the task:

1. Container Tilt Angle α. This represents the amount the pouring container is initially tilted to start the pouring.

2. Container Tilt Angle Speed ω. This represents the average speed used in tilting the container from the upright position to the final position.

3. Post Tilting Time t_p. This represents the time from the tilting completion to task completion.

4. Final Tilt Angle α_f. This represents the final tilt angle of the pouring container at task completion.

These parameters depend upon the task configuration parameters (p, v, f). The total task completion time is the sum of the tilting time and post tilting time.

The main steps behind our approach are described below:

Step 1 During the first phase of the approach, the human operator demonstrates how to do the task for many different settings of task parameters (p, v, f). We record both successful as well as unsuccessful demonstrations and score them appropriately. Each demonstration represents a point in the seven dimensional state-space $(p, v, f, \alpha, \omega, t_p, \alpha_f)$.

Step 2 We compute the transition boundary between successful and unsuccessful demonstrations. Each point on the transition boundary represents the values of $\alpha^*, \omega^*, t_p^*, \alpha_f^*$ for successfully doing the pouring task for the task configuration defined by (p, v, f).

Step 3 We learn simple rules that encode changes in parameters used by the humans that are used to go from unsuccessful demonstration to successful demonstration. For example, let $(p_u, v_u, f_u, \alpha_u, \omega_u, t_u, \alpha_{f_u})$ be a failed demonstration and let $\nabla_\alpha, \nabla_\omega, \nabla_{t_p}, \nabla_{\alpha_f}$ be the values of changes in parameters to transition from failure to success in the next demonstration, then we learn a simply rule that described how parameters should be changed to go from failure to success.

Step 4 Given a new task configuration (p, v, f), we compute task parameters $\alpha^*, \omega^*, t_p^*, \alpha_f^*$ using the transition boundary computed in Step 2.

Step 5 The robot executes the task using task parameters computed in Step 4. If the task is completed successfully, then the robot stops. If the task execution is not successful, then the robot finds the closest failed demonstration to task

configuration (p, v, f) from which human successfully transitioned to success. Parameters change rules associated with demonstration (computed in Step 3) are used to change the parameters and try again. This step is repeated until the robot succeeds in doing the task.

3.3.1 Generating Initial Task Parameters

Let D be the set of demonstrations performed by the human. Each demonstration $d \in D$ is represented as a triple (s, g, λ), where s is the state, g is the outcome (e.g., success, or failure), and λ is the score (e.g., merit score for success and penalty score for failure). Let D^s be the set of success demonstrations and D^f be the set of failure demonstrations. As discussed in the previous section, state is represented as $(p, v, f, \alpha, \omega, t_p, \alpha_f)$.

3.3.1.1 Human Demonstrations

A set of 190 human demonstrations of the pouring task was generated. Out of these, four outliers and 16 invalid trials (where the human performed multiple pours that do not fit into our task model) were removed. Accordingly, D used during the experimental evaluation had 170 demonstrations. This data was generated by observing four different human operators. Snapshots from a video recording of a sample human demonstration are shown in Figs. 3.2(a)-3.2(e). Each demonstration was labeled as either a success or a failure. Out of 170 demonstrations, 93 were

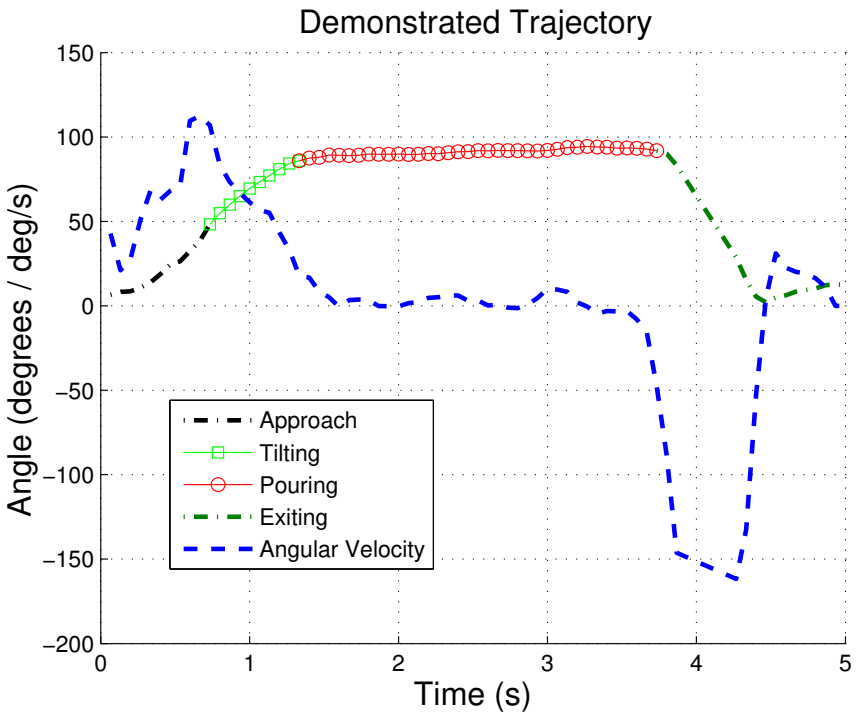

Figure 3.3: The demonstration trajectory extracted from the video shown in terms of the tilt angle of the bottle as a function of time.

labeled as 'success' and 77 were labeled as 'failure'. An appropriate score was assigned to each demonstration.

3.3.1.2 Parameter Extraction

Variables p, v, and f were set for each demonstration. Variables α, ω, t_p, and α_f were extracted automatically from video recordings of human demonstrations. Multiple visual tags are attached to both the pouring bottle and the table, which are detected by the Aruco augmented reality software library [161]. Figures 3.2(f) - 3.2(j) show snapshots from a video showing the visual tags detected by the tracking system for a human demonstration video (Figs. 3.2(a)-3.2(e)). Once the camera is calibrated for its position in the global coordinate system, the tag detection software

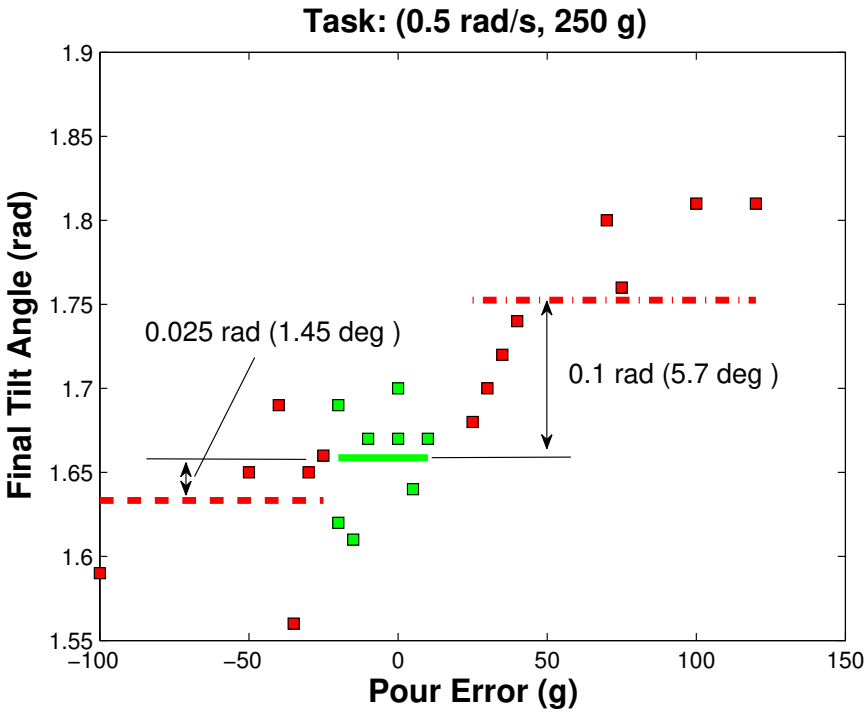

Figure 3.4: Plot of final tilt angle α_f as a function of pour error. A loose correlation can be seen between the parameter value and the trial performance.

can output both the position and orientation of the tags in global coordinates. The tag information was then used to calculate the angle that the bottle deviated from vertical. Other rotational information was ignored. This was done for each frame, generating a trajectory of the bottle's tilt over time. This trajectory was then numerically differentiated and smoothed by averaging nearby samples to provide an angular velocity trajectory as well. An example of both trajectories can be seen in Fig. 3.3.

To extract the pour task parameters, the trajectory was divided into four segments: (1) the approach to begin pouring, (2) the tilting phase, (3) the steady-state pouring phase, and (4) everything after pouring was finished. The segments were determined by thresholds on the trajectory, with tilting starting after the angle

exceeds 45 degrees (ϕ_{thres}), tilting ending when the angular velocity drops below 30 deg/s, and pouring ending when the angular velocity rises above 30 deg/s to return back to vertical. These separate segments can be seen, identified by color, on the previous figure. The initial tilt angle was then specified as the average of 3 samples after tilting ended. The tilt speed was the average of velocity samples during the tilt phase. The pour time was the duration of the pouring phase, and the final angle was the average of 3 samples prior to the pour ending. In all cases, multiple samples were taken to reduce the likelihood of noise in a single sample affecting the parameter value. Finally, the table speed was also directly measured by the tracking system, as one or two markers were always visible to the camera and their global coordinates at a table orientation of zero are known. After calculating the angle of the table from the visible markers for each frame, a line was fit to the data to obtain the speed.

3.3.1.3 Training a SVM Classifier

A support vector machine (SVM) is a maximum-margin classifier that can define a nonlinear decision using a kernel function. We begin by training a SVM on D. The SVM parameters and specific kernel function were selected by taking the best classification performance using 10-fold cross-validation.

3.3.1.4 Iterative Search

Given a new task configuration (p, v, f), the goal is to compute task parameters α^*, ω^* , t_p^*, and α_f^*. We generate many states by holding (p, v, f) constant and varying α, ω, t_p , and α_f. Currently we use 10 levels for each parameter. These lead to an initial set of 10000 candidate states $\mathcal{S}$. Each of the these initial states is classified as either a success or a failure using the classifier trained using D. We delete states that are labeled failure from $\mathcal{S}$. Let S^r represent the set of remaining candidate states. Now, we compute the closest distance from the states in S^r to success-states in D.

Let $s \in S^r$ and let s' be a success-state in D that is closest to s. We do iterative search on the line joining s and s' to find a success-state s" that is closest to s'. Any point on the line $\overline{ss'}$ is given by:

$$q(\gamma) = \gamma \begin{bmatrix} p \\ v \\ f \\ \alpha^s \\ \omega^s \\ t_p^s \\ \alpha_f^s \end{bmatrix} + (1-\gamma) \begin{bmatrix} p \\ v \\ f \\ \alpha^{s'} \\ \omega^{s'} \\ t_p^{s'} \\ \alpha_f^{s'} \end{bmatrix} \tag{3.1}$$

where $\gamma \in [0, 1]$. Accordingly, the search is performed by varying γ and using the classifier to check the status of states being generated during the search. From (3.1), note that variables p, v, and f remain constant for any value of γ. This step is performed for all states in S^r and the resulting such closest success-states are collected in the set S". Finally, we select s" $\in S$" that has the closest neighbor in D. This state is used to compute task parameters α^*, ω^*, t_p^* and α_f^*.

Algorithm 1 Algorithm to generate initial task parameters

Input: $S = \{s : s = (p, v, f, \alpha, \omega, t_p, \alpha_f)\}$,
    Human demonstrations:
    $D = \{(s_1, g_1, \lambda_1), (s_2, g_2, \lambda_2), \ldots, (s_n, g_n, \lambda_n)\}$, $n = |D|$
    $s_i \in S$, $g_i \in \{0, 1\}$ (0: failure, 1: success),
    $D^s = \{(s_i, g_i, \lambda_i) : g_i = 1\}$, $D^f = D - D^s$;
$kernel \leftarrow InitializeKernel(kernel\_name, kernel\_parameters)$;
svmStruct $\leftarrow$ $svmTrain(D, kernel)$;
Initialize new task configuration $(p, v, f) \leftarrow (p_0, v_0, f_0)$;
$\mathcal{S} \leftarrow$ $GenerateCandidateStates(\alpha, \omega, t_p, \alpha_f)$;
for $i = 1 : |\mathcal{S}|$ **do**
    $g_i \leftarrow svmClassify(svmStruct,\ s_i \in \mathcal{S})$
    if $(g_i == 1)$ **then**
        $s_i' \leftarrow \arg\min_{d_j \in D^s} ||s_i - d_j||$;
        $\gamma = 0$;
        while $(\gamma <= 1)$ **do**
            s_i" $\leftarrow q(\gamma)$; % From (3.1)
            g_i" $\leftarrow$ $svmClassify(svmStruct, s_i$")
            if $(g_i$" $== 1)$ **then**
                $ClosestDistance(s_i$") $\leftarrow ||s_i$" $- s_i'||$;
                break;
            end if
            $\gamma \leftarrow \gamma + \sigma$; % σ << 1 is a very small positive increment.
        end while
    end if
end forreturn $s^* \leftarrow \arg\min_{s_i"} \{ClosestDistance(s_i")\}$;

3.3.2 Refining Initial Task Parameters

The robot executes the task using the parameters α^*, ω^*, t_p^*, and α_f. Let s^* be the state associated with this task execution. If the task is successful, then we stop. If the task is not successful, then the robot needs to adjust these initial parameters.

For every unsuccessful demonstration in D, we ask the human operators as to what parameters they will change to improve the performance. This parameter is recorded for every unsuccessful demonstration. Let x be the parameter identified by the human with an unsuccessful demonstration d. We perform line search on this parameter using the SVM classifier described in the previous section to identify the minimum change in the value of the parameter to transition from failure to success. Let $\delta\bar{x}$ be the normalized value of target parameter change defined as:

$$\delta\bar{x} = \frac{\delta x}{x^f} \tag{3.2}$$

where x^f is the value of the parameter in the failed demonstration.

For every parameter that has been identified by human operators as a parameter that can be varied to improve the outcome, we develop an interpolation function that expresses the normalized value of the target parameter change as a function of the penalty score. The rationale for creating this function is that we expect a large change in parameter value to transition to success if the penalty score associated with the failed task is high.

We find the closest failed demonstration $d \in D$ to s^*. We use the parameter identified by the human operator in d for performing the change. We use the penalty score λ^* associated with s^* as an input to the normalized parameter change interpolation function to compute the change in the parameter. The robot tries again using the new parameter value. Currently, we stop after one round of parameter adjustment. In future implementation, this step will be repeated until the robot succeeds. Also, we have used a 5 DOF robot that does not match the human limb. So, these results may be even enhanced when using a more human-like robot arm such as the Baxter [162].

Algorithm 2 Algorithm to refine initial task parameters

Input: $s^* = (p^*, v^*, f^*, \alpha^*, \omega^*, t_p^*, \alpha_f^*)$,
svmStruct (Trained SVM classifier from Algorithm 1),
$D^f = \{(s_i, g_i, \lambda_i) : g_i = 0\}$,
human identified parameters for each failed state
$X = \{x_i : x_i \in \{\alpha, \omega, t_p, \alpha_f\}, i = 1, 2, \ldots, |D^f|\}$;
$D_1^f \leftarrow \{(s(x_i), 0, \lambda_i) : x_i (\in X) = \alpha\}$;
$D_2^f \leftarrow \{(s(x_i), 0, \lambda_i) : x_i (\in X) = \omega\}$;
$D_3^f \leftarrow \{(s(x_i), 0, \lambda_i) : x_i (\in X) = t_p\}$;
$D_4^f \leftarrow \{(s(x_i), 0, \lambda_i) : x_i (\in X) = \alpha_f\}$;
for $i = 1 : 4$ **do**
  $n_i \leftarrow |D_i^f|$
  for $j = 1 : n_i$ **do**
    while (1) **do**
      $x \leftarrow LinearSearch(x_{ij}^f)$;
      $g_j \leftarrow svmClassify(svmStruct, s(x))$;
      if $(g_j == 1)$ **then**
        $\delta \bar{x}_{ij} \leftarrow \frac{x - x_{ij}^f}{x_{ij}^f}$;
        break;
      end if
    end while
  end for
  $interp(i) \leftarrow polyfit\left(\{(\lambda_{ij}, \delta \bar{x}_{ij}) : j = 1, 2, \ldots, n_i\}\right)$
end for
$d \leftarrow \arg\min_{d \in D^f} ||s^* - d||$;
$i \leftarrow \arg_i (D_i^f : d \in D_i^f)$; % Index of failure subset that contains d. **return**
$x_i^r \leftarrow x_i^* \left[1 + polyval(interp(i), \lambda^*)\right]$;

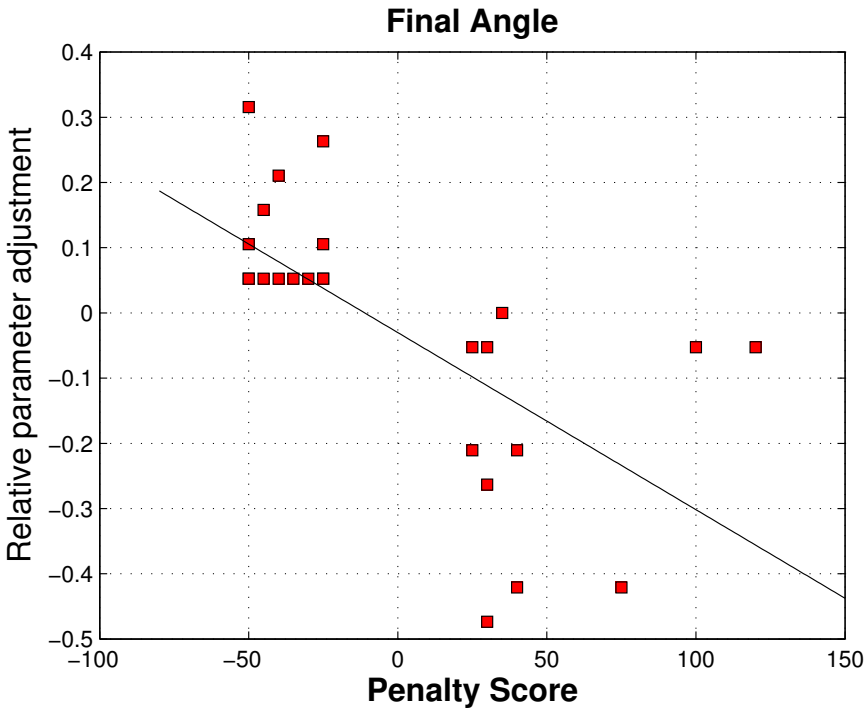

Figure 3.5: Interpolation function giving the relative necessary change of the parameter α_f as a function of the penalty score

3.3.3 Experimental Results

Implementation of our approach was conducted with a Labvolt Model 5150 robot manipulator and a rotating table of 40 cm radius. The robot has five degrees of freedom, allowing for full position control and control of the tilt angle of the bottle. For the pouring task, the position and speed of the table was measured using the same tracking system used for demonstrations. The arm trajectory was specified simply by solving the inverse kinematics to keep the opening of the bottle above the target container. Due to constraints in the robot's abilities, the pouring phase of the trajectory was required to be discretized into two segments of minimum 1.5 seconds each, which was still sufficient to vary the pour time and keep the bottle

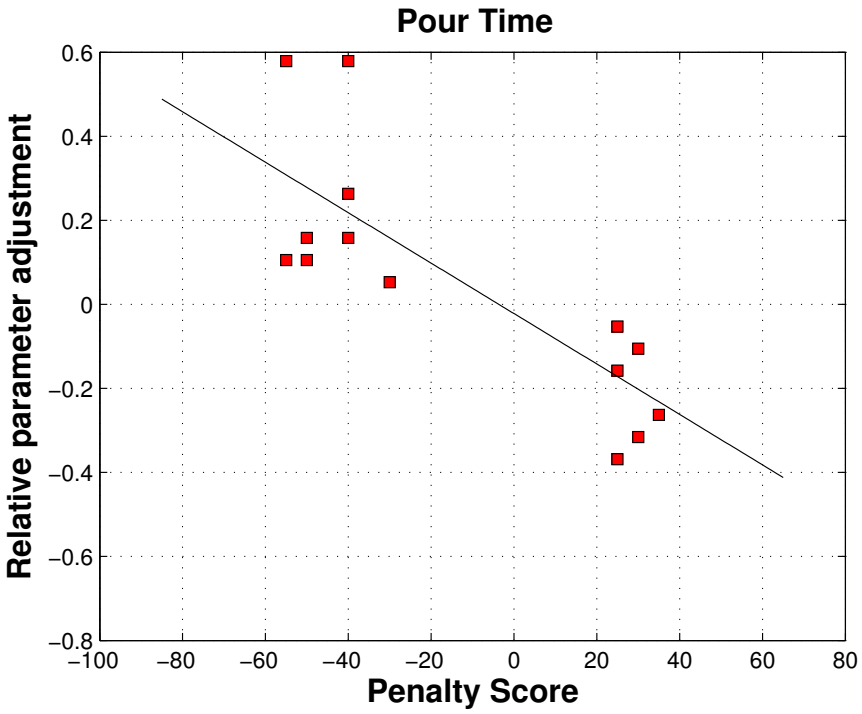

Figure 3.6: Interpolation function giving the relative necessary change of the parameter t_p as a function of the penalty score

above the target. The overall tilt of the bottle was not affected by the limits of the robot and the tilt profile was fully defined by the task parameters.

We wanted to generate parameters for the following task configuration: $p =$ 300 grams and $v = 0.34$ rad/s. We kept f (= 400 grams) fixed in all experiments). Using Algorithm 1 based on the approach described in Section 3.3.1, we computed the following task parameters: $\alpha^* = 1.4$ rad, $\omega^* = 1.28$ rad/s, $t_p^* = 4.1$ s, and $\alpha_f^* = 1.75$ rad.

Execution of the task with these parameters led to an over-pour of 325 g. For the closest data point in the failed demonstration set, the human had selected pour time t_p as the parameter to improve. The interpolation function (Algorithm 2) giving the relative necessary change of the parameter t_p as a function of the penalty

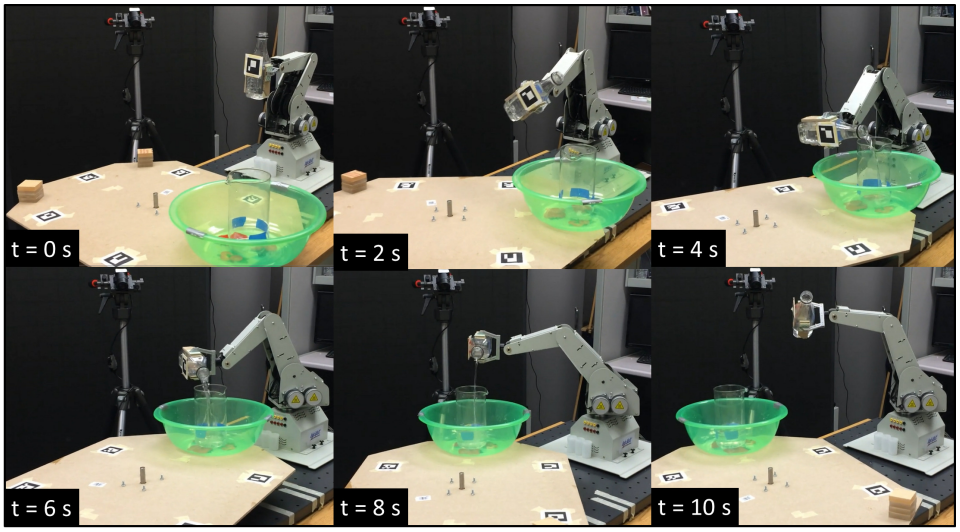

Figure 3.7: Snapshots from a video footage of the robot using the adjusted parameters to successfully perform the pouring task

score is shown in Fig. 3.6. Note that even with the noisy data, a physically realistic fit is achieved, with an over-pour leading toward a less pronounced pouring angle. It is also passes close to the origin, providing some qualitative stability assurance that small errors do not lead to large adjustments.

Then using t_p and the implied penalty score of +25, the derived interpolation function provided a relative parameter adjustment of -0.172. This provided a new pour time of 3.39 seconds. Execution of this updated set of parameters proved successful and the refinement process was halted.

3.4 Summary

An imitation learning approach was presented with the focus of allowing robots to learn from failed attempts by humans and how they recovered to achieve task success in dynamic manipulation tasks. A robot can use the learned models to

successfully perform in task scenarios which deviate from the ones demonstrated by humans. The presented approach uses an algorithm for finding new parameters to perform a task variant not previously seen by the robot or the human based on the success/failure classification of the points in the known dataset. A second algorithm uses feedback from the human after task failures to adjust parameter values and achieve task success after multiple iterations. These algorithms allowed the robot to successfully perform a dynamic pouring task under variations without a complicated planning or perception system.

The main contribution of this chapter is therefore an approach for dynamic tasks that can find successful parameters to new task variations using both successful and failed demonstrated task attempts. This approach can be used generally in robot learning of tasks that are computationally challenging to simulate and plan for, but are relatively simple for humans to perform, allowing the creation of a large library of demonstrations. This approach will also be highly effective when the demonstrator can easily specify the nature of any error they made during a failed attempt, providing that information for the system in its parameter adjustment phase.

However, there are some important limitations of this approach to note. As previously noted, the approach works well with a large demonstration library, with the dynamic pouring task using a relatively large set of 170 demonstrations to generate candidate parameters. This was feasible when each pouring trial took no more than a handful of seconds, but it may be excessively costly to collect this data for tasks of more complexity or longer duration. Additionally, the approach

required the demonstrator to be able to specify a single parameter to adjust after each failure. This requires the task to have a natural parameterization such that a human has an intuitive understanding of how each parameter affects the overall task output. For parameterizations that are automatically created or that may have a complex interaction of features, the demonstrator's intuition may actually provide misleading guidance for adjustments. Finally, when the robot begins using its own attempts to improve the initial model from demonstrations, it is mixing data points from systems with different dynamics, possibly running afoul of the correspondence problem. Therefore, it is useful to have a learning approach that does not need to rely on initial human demonstrations if the task is too complex. These factors were significant motivators for the work in the following chapter, which provides an approach for automatic parameter adjustments to achieve task success without a large initial set of demonstrations and without the strict need for a natural and intuitive parameterization.

Chapter 4: Selection of trajectory parameters for dynamic pouring tasks based on exploitation-driven updates of local meta-models

This chapter is partially derived from work presented at the IEEE/RSJ International Conference on Intelligent Robots and Systems: Workshop on Machine Learning in Planning and Control of Robot Motion, 2015 [163]

4.1 Introduction

Programming robots to perform real-world manipulation tasks is challenging and time consuming. For a large class of tasks, the robot arm must be provided with a precise trajectory to follow, prior to execution. For most currently deployed robots, these trajectories are defined manually by a skilled human operator. However, in the case of tasks with complex dynamics (e.g., manipulation of deformable materials and/or fluids), manual construction of analytical models that accurately capture the task dynamics is often not feasible, requiring a different method to generate the robots trajectory without resorting to trial and error on the part of the operator.

Such tasks often consist of a set of *instances*, with each instance defining a slightly different goal state and, therefore, the trajectory that the robot must follow. These different trajectories can be generated automatically, either through direct synthesis or by selecting the parameter values for a manually parameterized trajectory. In this chapter, we focus on the latter method of trajectory generation. Many tasks have a natural parameterization that a human can specify which can cover the full range of the robots ability. For example, a ball throwing task can be specified by having the arm move in a circular arc with a small set of parameters such as the radius, speed, and release point fully controlling the throwing distance. We are interested in the problem of how the parameter values can be determined for each task instance automatically in order to generate the final trajectory to be executed by the robot.

A common approach taken to find the parameter values for a particular task instance is to use a model of the task and search for valid parameter values using the predictive ability of the model. To obtain the highest prediction accuracy, it is typically ideal to use a physics-based simulator that can capture the details of most, if not all, of the task dynamics. However, for complex tasks, such simulators are either unavailable or generally far too computationally expensive to be able to use for parameter selection in reasonably short time periods. This is especially true for tasks involving deformable materials and fluids, which would require full finite-element simulations. Instead of simulators, such tasks can be modeled with surrogate models created from the data that the robot collects as it gains experience while attempting the task. This framework requires the robot to go through a learning

period where it may perform task instances incorrectly until it collects enough data to find valid parameter values for all the desired instances.

An important aspect of the types of tasks we address, which is reflected in our approach, is that many satisficing parameter values are available for a given task instance. We do not address the problem of finding the single optimum set of parameters or searching for valid parameters when they are clustered together in small regions of the space. Instead, our approach is designed for tasks where valid parameter sets are found throughout the space and it is sufficient to find a single one. The focus is to exploit the available knowledge to find satisficing parameters with as few task attempts as possible.

Our proposed approach has the goal of rapidly finding valid solutions in the parameter space corresponding to new task variations with sparse initial data. We model the task abstractly as a set of parameters existing in a finite-dimensional space where each point in the space defines a trajectory to perform a single task variation. First, in a *model generation* phase, local models are constructed in the vicinity of the previously conducted experiments that explain both the task function behavior and the estimated divergence of the generated model from the true model when moving within the neighborhood of each experiment. Second, in an *exploitation-driven updating* phase, these models are used to guide parameter selection given a desired task outcome and the models are updated based on the actual outcome of the task execution. Our approach exploits local information available in adaptively chosen neighborhoods effectively to incrementally build and update multiple local models, thereby allowing the algorithm to capture arbitrarily complex function landscapes.

We validate our approach by testing it both on synthetic nonlinear functions and on a physical robot. For physical experiments, we consider a dynamic pouring task, in which the robot is tasked to pour a certain volume of liquid from a bottle it is holding into a container placed on a rotating turntable. Moving the bottle to track the moving container while simultaneously tilting it to pour creates complex fluid dynamics. This makes the mapping from pouring-trajectory parameters to poured volume highly nonlinear and infeasible to model analytically. Our results reveal that the correct pouring parameters for a new pour volume can be learned quite rapidly, with a small number of exploratory experiments.

4.2 Problem Formulation

4.2.1 Problem Statement

In this chapter, we utilize a framework where a robot performs a task by executing a finite-length trajectory τ. The task is characterized by an unknown dynamics function that produces a real-valued output, or score, for a particular trajectory:

$$y = f(\tau), \quad y \in \mathbb{R}. \tag{4.1}$$

Note that this task function is different from typical robot dynamics functions that only map the current state to either a future state, or the derivative of the state. Here, the task output is a function of the entire trajectory.

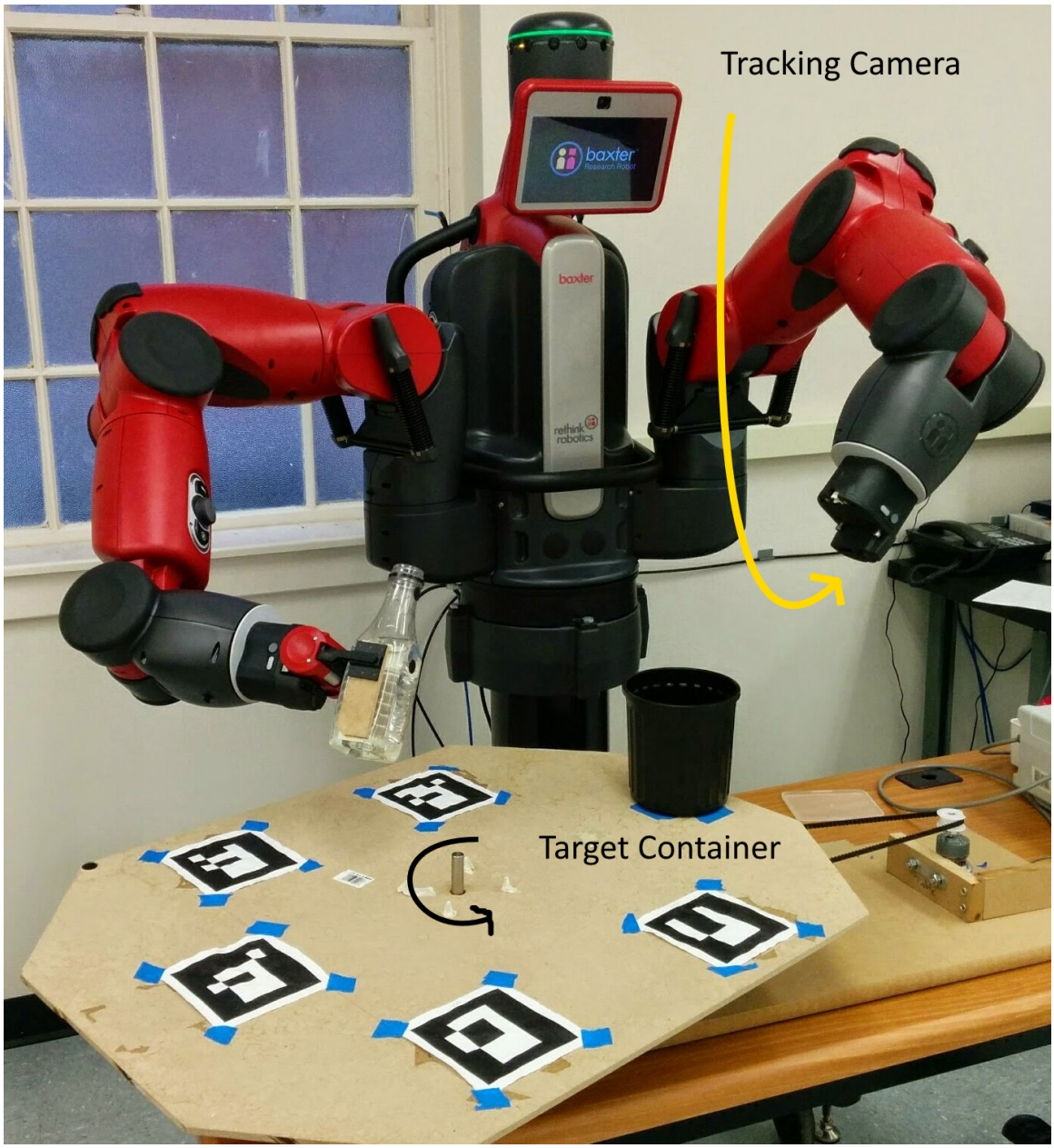

Figure 4.1: Experimental setup used for the pouring task. The right hand holds the bottle and performs the pouring while the camera in the left hand monitors the visual markers on the table to measure its current position and speed.

Rather than learn the trajectory directly, we assume the trajectory to be the output of a task-specific planner π with an input of a parameter vector θ of length p. We assume that the structure of the planner can be defined manually, leaving the detail of producing the correct task output to the learning algorithm, which will find the corresponding parameter values. For example, in a task such as throwing a ball a certain distance, a planner might be specified to move the robot end-effector in a circular arc. This trajectory may comprise parameters like radius, speed, and release point, the values of which determine the final throwing distance. This method can enable faster task learning by combining an intelligent human task parameterization with an automated learning algorithm. The algorithm then operates on the direct mapping from task parameters to outputs, which is the composition of the planner with the task dynamics function:

$$\tau = \pi(\theta), \quad \theta \in \mathbb{R}^p$$

$$y = h(\theta) = (f \circ \pi)(\theta) = f(\tau)$$

The overall task itself is represented by a set of task *instances*, each describing a specific desired task output value T_i. Each task instance has a corresponding non-negative cost function C_i and tolerance amount δ_i such that, if y_i is a successful output of the i^{th} instance, then $C_i(y_i) < \delta_i$. For the remainder of the chapter, we consider the simple cost function $C_i(y) = |y - T_i|$. When the robot is assigned a specific task instance, it conducts *attempts* to solve the instance. Each attempt con-

sists of selecting the parameters, executing the resulting trajectory, and measuring the resulting cost function.

The problem addressed here can be formulated as follows: Specify a computational procedure Ψ that the robot will execute to solve a task instance, given the knowledge acquired from previous instances and attempts. The procedure consists of an evaluation loop where a candidate parameter vector is determined from m previous attempts and the corresponding trajectory is executed. The previous attempts ($\Theta_m = \{\theta_1, \ldots, \theta_m\}$, $Y_m = \{y_1, \ldots, y_m\}$) may come from previous instances, previous attempts of the current instance, or prior knowledge obtained elsewhere such as random samples. If the corresponding task output for the candidate is successful, the procedure terminates. Otherwise, the procedure will iterate with a new candidate selected after adding the previous attempt to Θ_m and Y_m.

$$\theta_{m+1} \leftarrow \Psi(T_i, \Theta_m, Y_m) \tag{4.2}$$

In general, the user will assign not just a single task instance during the robot training, but will be interested to have the robot learn a set of many different instances. We assume that such a set is not known beforehand and may possibly not even have a known final size when the robot begins to learn the first instances. Without knowledge of future instances to learn, the system focuses on each instance separately, with each one being solved before addressing the next.

In this framework, we desire to find a procedure that can solve each instance with a minimal number of attempts. Therefore, we use the total number of attempts made to solve N_t task instances as the overall cost function:

$$J(T) = \sum_{i=1}^{N_t} \eta_i, \tag{4.3}$$

where $T = \{T_1, \ldots, T_{N_t}\}$, and η_i is the number of attempts taken to solve the i'th instance. Since the task instances are solved independently, the ideal procedure can directly minimize each η_i without regard for the other instances.

While the optimal procedure that produces $\eta_i = 1$ for all instances is impossible to acheive, we do aim to specify a procedure that can asymptotically reduce η_i to 1, while simultaneously trying to minimize the number of attempts for the earlier instances. In practical robotics scenarios, it is important to explicitly address the cost of learning all task instances as N_t must be small enough so that the complete learning is feasible and the costs for the early instances cannot be discounted.

4.2.2 Evaluation of Existing Approaches for Application to Trajectory Parameter Selection Problem

Before explaining our approach in detail, it is useful to briefly discuss how existing techniques can be applied to this particular problem and what are some of the problems they experience. We discuss two major alternatives for addressing our problem and provide illustrative comparative results for two representational algorithms.

4.2.2.1 Reinforcement Learning Based Policy Search

Our problem can be posed to work under the standard reinforcement learning framework used in robotics. The framework is composed of unknown system dynamics, which are influenced by a controller, whose output is determined by a parameterized control policy.

$$\begin{aligned} x_{t+1} &= f(x_t, u_t) \\ u_t &= \pi_\theta(x_t) \end{aligned} \tag{4.4}$$

A cost function $J(\theta)$ is defined for the policy parameters, which is evaluated over one or more trajectories followed by the system with a controller executing the policy. The goal of any learning algorithm is to find the set of policy parameters θ^* that minimize the cost function.

$$J(\theta) = \sum_{i=1}^{T} c(x_i) \tag{4.5}$$

While superficially different, the formulation of our problem in the previous section can be considered a special case of the general RL framework. The state space is taken as simply the task score itself, $z \in \mathbb{R}$. Instead of a multi-step trajectory determined by the system dynamics, we can consider just a single step from an initial state to the final output. The control policy is taken to output the task parameters

themselves. Dropping the initial state from the model, as it is always the same, these simplifications reduce the general system dynamics in (4.4) to match (4.1) in section 4.2.1.

4.2.2.2 Adapting the PILCO Algorithm

The PILCO algorithm [22] provides an approach to simultaneously learn the system dynamics and find preferred policy parameters for the general reinforcement learning framework. The core of the algorithm is to approximate $f(x, u)$ with a Gaussian process regression model, where the current state and control input are used as model inputs and the future state (or change in state) is output as a prediction. The GP outputs a probability distribution for its output, which can be propagated through the system dynamics in (4.4). This allows the algorithm to compute an expression for the expected cost of following a particular policy, with the current knowledge of the system dynamics. To improve the policy, the derivative of the cost function with respect to the policy parameters can be directly computed, with slight assumptions on the structure of the policy and cost function. The derivative allows standard local optimization algorithms, such as the BFGS method, to converge to a local optimum for the policy parameters.

When adapted to our problem, the complex computations used in PILCO for the general case are dramatically simplified. With only a single state update, the cost function only depends on a single output for the GP, which is a normal distribution. The expected value of this output is therefore just the mean of the

Algorithm	Mean # iters	% within 10 iters
Adapted PILCO	12.7	48%
Local linear models	5.6	90%

prediction and using a simple squared-error state cost function $C(z) = (z - z^*)^2$, the policy cost reduces to

$$J(\theta) = (\mu(\theta) - z^*)^2. \tag{4.6}$$

Computing the derivative of $J(\theta)$ is then straightforward, given a smooth choice of covariance function for the GP.

This adapted formulation of the PILCO approach was compared against our current method of searching for task parameters, using a set of local linear models, on a synthetic 5-dimension nonlinear test function (See Sec. 4.5 for a full description). For both experiments, the same initial set of 50 random task parameter samples and their corresponding outputs were given to the learning algorithms. One hundred target outputs, uniformly distributed over the range of the test function, were given individually to the algorithms, with the known data set reset to the initial 50 samples after each target was found. For the PILCO policy search, the initial parameters were set to the mean of the initial data set. Searching with linear models does not require any initialization.

Primarily due to local updating, the adapted PILCO policy search has noticeably worse performance than our approach using linear models search as seen in Table 4.2.2.2, which shows the results of 100 different target searches. For PILCO, out of 100 targets attempted, 40 could still not be found after 20 samples of the

test function, most likely indicating that the policy parameters became stuck in local minima, which is something our approach is not vulnerable to. To conclude the comparison, when the learning problem is formulated in the framework we have developed our approach for, existing reinforcement learning techniques, represented by PILCO, address the problem in a simplified manner. Reducing the problem to performing only local updates of the policy parameters leads to much worse performance than can be achieved by our proposed approach.

4.2.2.3 Bayesian Optimization Framework

An alternate approach to solving the problem is with an optimization framework. For a desired task score z^*, the same cost function for an attempt in the reinforcement learning can be used:

$$C(\theta) = (z - z^*)^2 = (f(\theta) - z^*)^2. \tag{4.7}$$

A search algorithm can then attempt to minimize this cost function to find a set of task parameters that provide the desired task score. The algorithm terminates when a minimum value within the task tolerance of zero is found.

A Bayesian optimization framework allows an algorithm to select the next best point to test in the space based on a current probability model which consists of a prior that is conditioned on information acquired during the search. A Gaussian process is a straightforward model that can be used in this Bayesian manner.

4.2.2.4 GP-UCB Algorithm

Given a probability model conditioned on known information about the task, the question arises of how to select the next point to test, which requires some trade-off of exploration and exploitation. A simple algorithm that balances these factors is the Gaussian Process Upper Confidence Bound algorithm [164]. Essentially, given a set of candidate points D where the next test point can be taken, the candidate selected is

$$\theta_{m+1} = \underset{\theta \in D}{\operatorname{argmax}}\, \mu_m(\theta) + \beta_m^{1/2} \sigma_m(\theta), \tag{4.8}$$

where $\mu_m(\theta)$ and $\sigma_m(\theta)$ are the mean and the standard deviation of the Gaussian process at θ when conditioned on m previous samples. The weighting parameter β_m is set to $2\log(|D|m^2\pi^2/6\delta)$ for a user chosen value of δ, which allows for provable bounds on the regret incurred during the optimization process. In practice, however, β_m can be tuned for faster convergence, which is detailed in the previous citation.

This algorithm was adopted for our problem and cost function, flipping the signs in (4.8) for minimization. We also did not use a fixed candidate set D, but rather used (4.8) with an off-the-shelf optimization library which searched for the minimizing value within the valid parameter bounds. We discovered that our adaptation of GP-UCB was effective in quickly searching the parameter space but had substantial problems finding solutions with very high precision. This would happen because while the algorithm quickly found regions when zero-error parameters

Algorithm	Mean # iters	% within 10 iters
GP-UCB	22.1	30%
Local linear models	2.0	95%

were nearby, it had little incentive to search closeby to the newly tested points to lower the error further. This is due to the uncertainty decreasing to near-zero in the region of the new test point, while other regions have much higher uncertainty, and are therefore weighted higher by the GP-UCB heuristic. To get decent performance with the search we had to loosen the task tolerance considerably but this only improved the performance of our own approach as well, which was far better. This distinction is shown in Table 4.2.2.4, which contains the results from 20 target searches while using a very high task tolerance value. Given this behavior, we concluded that using a standard optimization method was not ideal for this problem as (1) we were trying to find parameters where the objective function was exactly zero and not just minimal, and (2) information about where zeros may lie was being discarded by using a squared error term instead of retaining the sign.

4.2.2.5 Observation

To conclude this section, we observed that our particular problem has characteristics that make it challenging for existing methods that are broadly applicable. Since our task involves only a one-step update directly to the final score, the full power of reinforcement learning algorithms cannot be made use of, and they are reduced to simple local updates of the policy parameters. And because we are searching for parameters that are exactly zeros of a non-negative cost function, a

Bayesian optimization algorithm is not ideal, as it moves to other more uncertain regions of the parameter spaces once it learns that a particular area is not going to go any lower than zero. These difficulties will be explicitly addressed by the features in our approach to greatly improve the learning performance for this particular type of problem.

4.2.3 Overview of Approach

This chapter presents an approach and procedure to find solutions to new task instances by iteratively improving a model of the task dynamics function $h(\theta)$. Since the individual task instances are presented to the robot sequentially, and it is not known a priori what the total number will be, we adopt the overall minimization strategy of solving each instance with as few attempts as possible. However, to reduce the number of attempts for possible later task instances, the learning algorithm continually gains experience of the task dynamics in order to exploit available knowledge and find faster solutions.

An important property of tasks that can learned efficiently with this strategy is that many solutions exist in the parameter space for each task instance, and hence can be found easily. The fact that large solution sets exist throughout the space allows a strategy of selecting test points by adjusting a single parameter value. This helps to maximize the information gained through multiple attempts and find solutions rapidly. If the solution set is instead sparse and clustered into small regions, using single parameter adjustments is likely to perform worse, as it

becomes less likely that a solution exists somewhere on the line obtained by allowing a single parameter to be free. Many robot tasks have natural parameterizations, however, that show this behavior of being able to find solutions by adjusting a single parameter only.

The general outline of the approach involves the following components. First, at each known sample point, a neighborhood is defined and used to compute a local linear model to approximate the function behavior. Second, an error model at each point is computed to approximate the region where the linear model is considered accurate. Candidate parameter adjustments which are expected to solve the task instance in one step are generated from all current sample points, and are compared to select the one with the highest confidence. The new point in the parameter space is sent to the robot for execution and if the resulting task output is not a success, the system repeats the full strategy with the new information.

4.3 Preliminary Experiments using Existing Regression Algorithms

In this section, we test GPR and LWPR on regression and inverse regression tasks and investigate how they perform when the training data is sparse. Given a training data set comprising samples of inputs (explanatory variables) and the corresponding output (dependent variable), the *regression* task consists of learning the input-output mapping and using the learned model to provide an estimate of the output for any point in the input space. Conversely, given a target (desired output

value), the *inverse regression* task consists of finding points in the input space that map to function value within some specified tolerance of the desired output.

4.3.1 Gaussian Process Regression Method

GPR is a global nonparametric algorithm that can make highly accurate predictions of function values after being trained on sufficient data. Details on the training and prediction algorithms are discussed in Appendix A. As mentioned previously, the primary disadvantage of using a GPR model is that when new data is acquired, the model must be retrained on all existing data, making the computation cost prohibitive for very large amounts of data. The number of samples in our comparisons, however, was small enough that this was not a significant obstacle. Additionally, it is important that the hyperparameters of the Gaussian process are set to reasonable values for the problem space, most significantly the kernel parameters that control the expected output similarity of nearby points in the input space. We used a standard expectation maximization algorithm, as described in Appendix A, which performs a gradient ascent of the expectation function in the hyperparameter space. This computation, because it is iterative, is expensive enough that reoptimizing the hyperparameters each time after new data is acquired is infeasible. Instead, we found parameter values from the initial data set and continued to use them throughout the learning process.

4.3.2 Locally Weighted Projection Regression Method

LWPR uses an entirely different modeling approach of locally capturing the target function behavior with linear models. LWPR was designed for efficient incremental learning with high dimensional data.

The core of LWPR is to continually place and adjust linear models throughout the data space which cover the full domain of the function to be learned. Each linear model, however, does not have the full dimensionality of the data but rather, uses partial least squares to detect which directions in the input space are relevant to the function behavior. This allows the model to learn fewer parameters than a more brute-force approach, enabling efficient incremental learning ability. Furthermore, the locations, distance metrics, and number of linear models are all adjusted by the learning algorithm as more data is available.

The local form and parameters of the model can be seen in the primary function prediction equation, which is a weighted average of the local models:

$$f(\mathbf{x}) = \frac{\sum_{k=1}^{K} w_k(\mathbf{x})\Psi(\mathbf{x})}{\sum_{k=1}^{K} w_k(\mathbf{x})} \tag{4.9}$$

The weights on the models are calculated using a Gaussian kernel function parameterized by a center position, $\mathbf{c}_k$, and distance metric, D_k which controls the activation value of test points.

$$w_k(\mathbf{x}) = exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{c}_k)^T D_k(\mathbf{x}-\mathbf{c}_k)\right) \tag{4.10}$$

The models themselves are linear, projected into a lower-dimensional space as determined by the behavior of the target function in the vicinity of the model center. The linear model and projection equations are:

$$\Psi_k(\mathbf{x}) = \beta_0 + \sum_{i=1}^{R} \beta_i s_i \tag{4.11}$$

$$\begin{aligned} s_i &= \mathbf{u}_i^T \mathbf{x}_{i-1} \\ \mathbf{x}_i &= \mathbf{x}_{i-1} - s_i \mathbf{p}_i \\ \mathbf{x}_0 &= \mathbf{x} - \bar{\mathbf{x}} \end{aligned} \tag{4.12}$$

The parameters R, β_0, β_i, u_i, and p_i are computed statistically and on-the-fly from incoming data. More specific details on the implementation and methodology are available in [87].

For our purpose, we trained the LWPR model by incrementally updating the model by providing a single point at a time from the presampled data set. The LWPR authors suggest that the model performance suffers when only sparse data is available, as in our case. To alleviate this limitation and to allow the model to converge to stable prediction values, the data is fed to the learning algorithm multiple times in random order, per the advice of the authors.

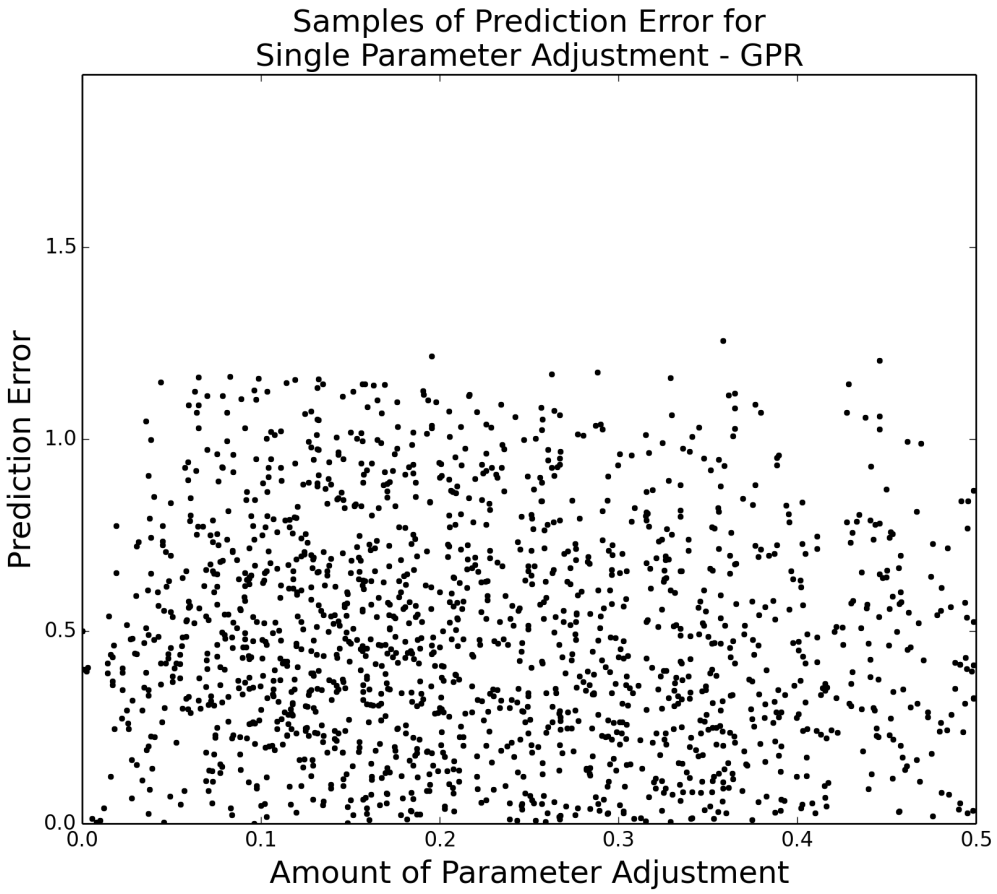

Figure 4.2: Performance of GPR on the inverse regression task: modeling error when attempting to predict parameter values that correspond to a desired target is plotted. The model was trained on the same sparse data set (50 samples) and 50 targets were defined over the function range. Each point represents a sample where a single parameter value was adjusted from one of the known points until the predicted function value was within tolerance of the target. The y-axis shows the error of this prediction from the true function value.

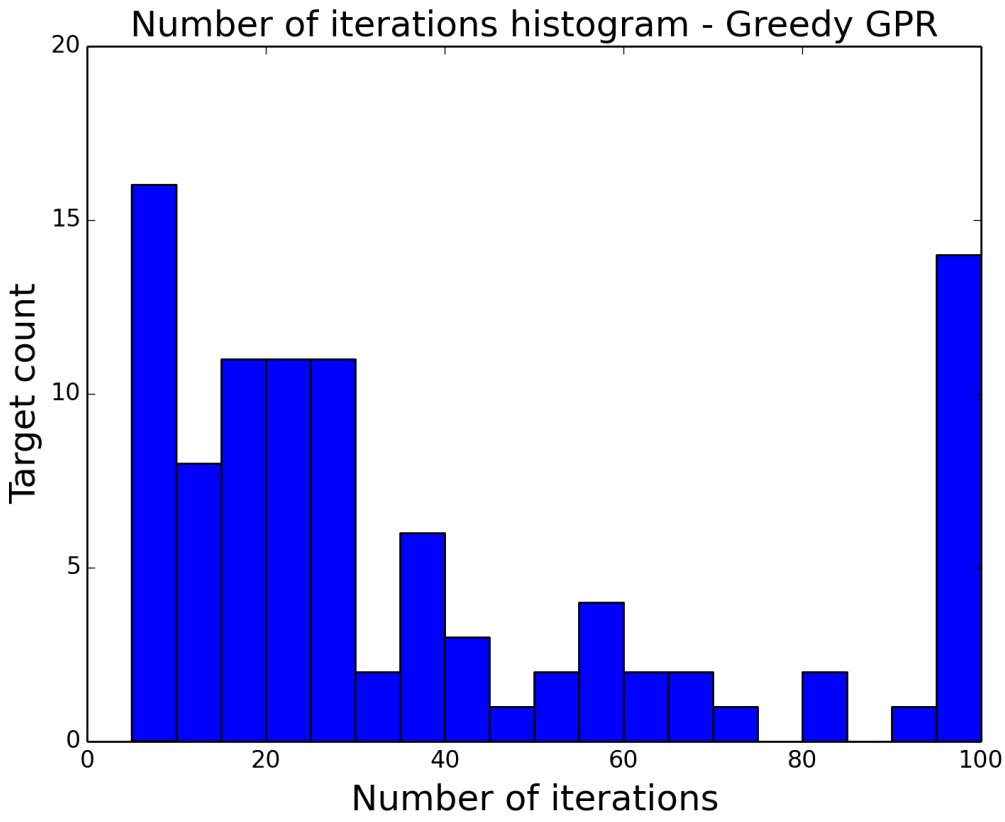

Figure 4.3: The performance of the GPR algorithm on the greedy search task. 100 target function values were given to the algorithm with along with an initial library of 50 random samples. The library was reset to the same initial 50 points for each target. The algorithm was set to time-out at 100 iterations and the targets that had not yet been found (13) are collected in the bin at 100.

4.3.3 Modeling Comparison

First, we compared GPR and LWPR in terms of their regression capabilities. The metric used for model prediction accuracy was the average squared-error of the predicted values, sampled randomly over the function domain. The 5D Schwefel function was used in these experiments. GPR had much better performance for all tested data set sizes, with a slightly decreasing advantage as more samples are available. As LWPR achieves its efficiency by collapsing the training data into a smaller set of linear models, it would be expected that retaining all of the training data for prediction as GPR does would increase its accuracy considerably.

Next, we considered the following inverse regression problem: Given a target (desired function value), find points in the parameter space that map to a function value within a specified tolerance of the given target. Figure 4.2 shows samples of GPR modeling error when attempting to predict parameter values that correspond to a target. The model was trained on the same sparse data set (50 samples) and 50 targets were defined over the function range. We were mainly interested in finding how the modeling error varied as a function of distance from a sampled point along each single parameter direction. Accordingly, in the figure, each point represents a sample where a single parameter value was adjusted from one of the known points until the predicted function value was within tolerance of the target. The y-axis shows the error of this prediction from the true function value. Note from the figure that there is no pattern in the modeling error as a function of distance from a known point.

We tested a simple greedy strategy in which, given a target score, the parameter space was searched with a standard optimizer to find a point where the solution (set of parameters that mapped to a function value within a specified tolerance w.r.t. the target value) was reached with maximum confidence. Figure 4.3 shows the results of GPR based greedy search strategy on the five-dimensional Schwefel function. A set of 100 target function values was given to the algorithm along with an initial training dataset of 50 random samples. The dataset was reset to the same initial 50 points for each target. As the task was to find points on a synthetic function with no noise, σ_n^2 was set to a very small value (10^{-6}), though still nonzero for numerical stability.

Note from the figure that the algorithm finds solutions for a minority of target values (19%) after a handful of iterations (≤ 10). However, there is a sizeable range of the distribution with the algorithm taking 10 to 50 iterations (53% of targets) and a minimum of 100 iterations for some targets (13%) before a solution is found.

We were unable to get good results from LWPR on the inverse regression problem for sparse data sets. We had to boost the data sizes up to 1000 for LWPR to work properly. For very small data set sizes, LWPR requires the samples to be given to the algorithm repeatedly in random order until convergence. We discovered that when only a few samples are seen repeatedly, the algorithm would mostly learn local models that were influenced by only a single point and were flat (had zero slope) in the parameter space, essentially modeling the function in a simple nearest-neighbor manner. As inverse regression with a flat plane is an ill-posed problem, we were unable to generate the LWPR equivalent of the search result in Fig. 4.3.

Summarizing our observations during these preliminary experiments, LWPR performs much worse than GPR for sparse data sets and in the case of GPR, in general, many task executions are needed before the solution is reached.

However, in this work, we would like to attempt problems where such training data is not already available or is very expensive to obtain. Accordingly, our primary goal is to efficiently exploit the available information and achieve convergence to a solution with as few task executions as possible and not necessarily explore the parameter space to build a globally accurate model.

4.4 Local Linear Metamodel Based Approach

The primary guiding principle of the proposed approach is to iteratively search the task parameter space by creating local models of parameteric neighborhood and finding a movement within a carefully chosen neighborhood that has the minimum uncertainty. First, in an initialization phase, a set of training samples is obtained from either human demonstrations or robot executions of randomly selected points in the parameter space.

Each sample is a tuple comprising a parameter vector and its mapping to a task score. Second, in a *model generation* phase, the algorithm builds local models based on the current training set. Third, in an *exploitation-driven model updating* phase, the algorithm finds a new point by using the current set of local models, evaluates its task score (for example, in the context of the robot pouring task, this corresponds to the robot using the found tilt parameters to execute the task and

measuring the poured amount), adds the newly found point to the training set, and updates the set of local models. The cycle of model generation and model exploitation repeats until a point is found whose task score is within the success tolerance of that of the desired task.

4.4.1 Initialization

Let $\mathcal{S} = \{(x^{(i)}, y^{(i)}) : x^{(i)} \in \mathcal{X}, y^{(i)} \in \mathcal{Y}, i = 1, 2, \ldots, m\}$ be the initial set of training samples, where $\mathcal{X}$ is the set of sample points in the parameter space, $x^{(i)} \in \mathbb{R}^n$ is the i^{th} point in $\mathcal{X}$, $\mathcal{Y}$ is the set of corresponding task scores, $y^{(i)} \in \mathbb{R}$ is the i^{th} task score in $\mathcal{Y}$, and m is the number of training samples. Let $x_j^{(i)} \in \mathbb{R}$ represent the j^{th} element of $x^{(i)}$. Further, let $x \in \mathbb{R}^n$ represent a general point in the parameter space.

4.4.2 Model Generation

This phase is achieved in three steps: (1) adaptive neighborhood selection, (2) planar model approximation, and (3) error-divergence model approximation.

4.4.2.1 Adaptive neighborhood selection

For each point $x^{(i)} \in \mathcal{X}$, the algorithm builds a local linear model by using points $x^{(j)} \in \mathcal{X}$ residing within an adaptive box-neighborhood $\mathcal{N}^{(i)} \subset \mathcal{X}$:

$$\mathcal{N}^{(i)}(k) = \{x^{(j)} \in \mathcal{X} : ||x^{(i)} - x^{(j)}||_\infty \leq \delta_k^{(i)}\} \tag{4.13}$$

where, k is the number of neighbors in the neighborhood and $\delta_k^{(i)}$ is the neighborhood size, which is given by

$$\delta_k^{(i)} = \max_{x^{(j)} \in \mathcal{N}^{(i)}(k)} ||x^{(i)} - x^{(j)}||_\infty \text{ s.t. } |\mathcal{N}^{(i)}(k)| = k \tag{4.14}$$

According to (4.13) and (4.14), note that $\delta_k^{(i)}$ is assigned to the minimum possible size that results in $k-$nearest neighbors. Since the density of points in the data set can be highly variable, it is important to find a $\delta_k^{(i)}$ that results in a neighborhood with sufficient points to generate a relatively accurate local approximation, but not so many that the nonlinear behavior of the underlying function deteriorates the approximation. This is done by using a leave-one-out cross-validation technique to estimate the optimal neighborhood size $\delta_{k^*}^{(i)}$. In particular, for each $x^{(j)} \in \mathcal{N}^{(i)}(k)$, a plane is fit using least-squares on the set $\mathcal{N}^{(i)}(k)/x^{(j)}$ and the linear-fit error at $x^{(j)}$ is computed. Now, the mean of linear-fit errors η_k over all $x^{(j)} \in \mathcal{N}^{(i)}(k)$ is used as a fitness to evaluate the neighborhood size.

We consider $k = n+1$ as the least number of desired points in $\mathcal{N}^{(i)}(k)$ since n points are needed for a unique plane fit, plus an additional point for cross-validation error measurement. Accordingly, the neighborhood size is initialized to $\delta_{n+2}^{(i)}$ and the corresponding η_k is computed. Next, k is incremented by one and η_{k+1} is computed. If $\eta_k < \eta_{k+1}$, then $\delta_k^{(i)}$ is reported as optimal. Otherwise, the search continues to find a better neighborhood-size.

In general, if sample density around $x^{(i)}$ is moderate, the successive error values will decrease as the plane fits are less sensitive to noise induced from few samples,

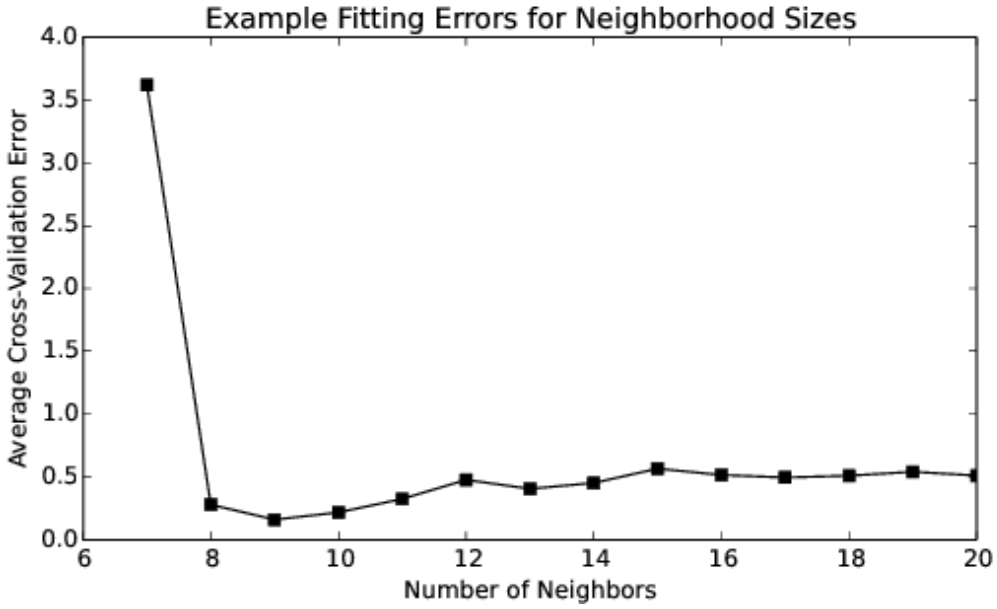

Figure 4.4: An example of neighborhood cross-validation error fits as the neighborhood size increases. This particular example behaves nicely and the algorithm will select the optimal size at N=9.

but then increase again as the neighborhood begins to include the nonlinearity of the sampled function. Finding the neighborhood-size when the error first increases is used as a rough heuristic to find the balance between these two factors while also not evaluating more neighborhood sizes than necessary. Figure 4.4 shows an actual computed example where this heuristic happens to give the optimal neighborhood size.

Figure 4.5 shows the resulting behavior of this algorithm, which is to shrink neighborhood sizes in regions of high sample density so as to maintain model accuracy. This algorithm does generate similar neighborhood sizes as the simpler heuristic of just using the nearest N neighbors, but, as will be shown later, accounting for the accuracy of the plane fit leads to better target search results.

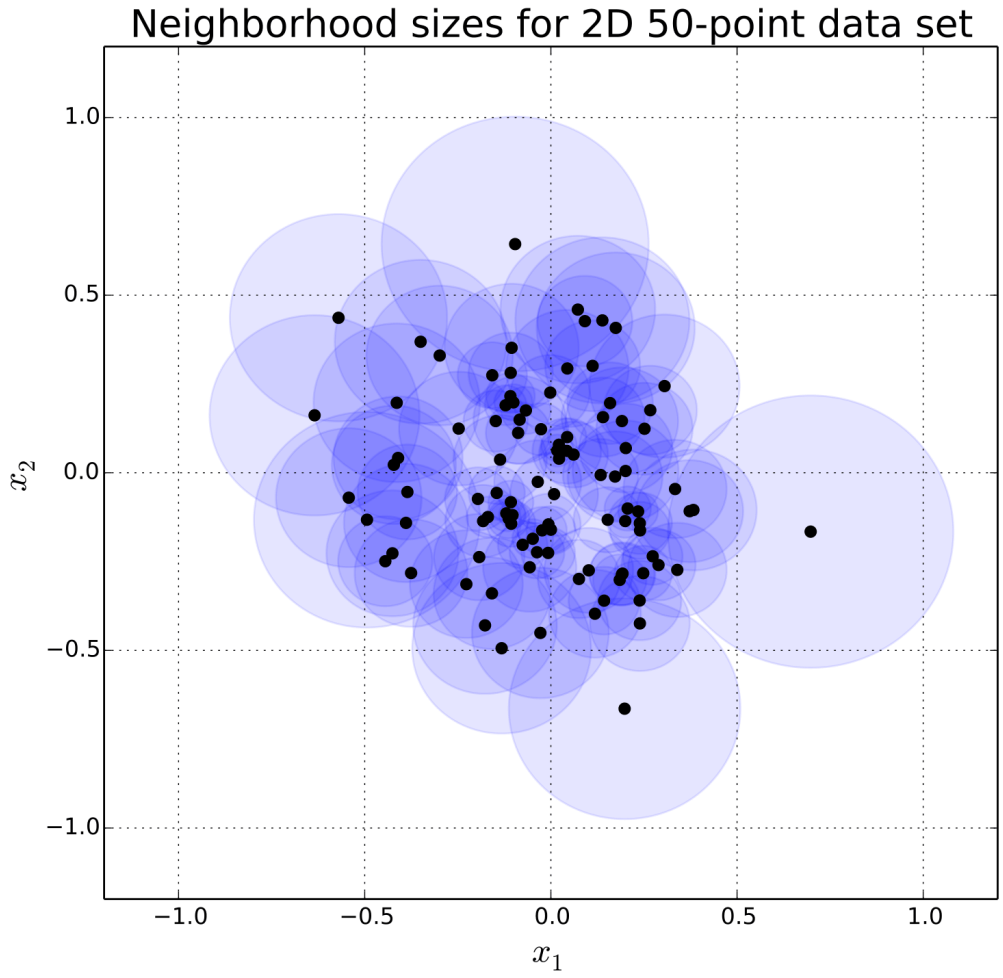

Figure 4.5: Example neighborhood sizes computed using the adaptive heuristic for normally-distributed 2D data.

4.4.2.2 Planar model approximation

An affine hyperplane $\mathcal{F}^{(i)}(x)$ is fit to points in the neighborhood $\mathcal{N}^{(i)}(k^*)$, obtained from the previous phase, by using a least-squares method:

$$\mathcal{F}^{(i)}(x) = A^{(i)}(x - x^{(i)}) + y^{(i)} \tag{4.15}$$

where, $A^{(i)} = \begin{bmatrix} a_1^{(i)} & a_2^{(i)} & \cdots & a_n^{(i)} \end{bmatrix}$ is a row vector of planar model coefficients. If the plane is not uniquely determined (e.g., all the points are collinear), then the neighborhood size is incrementally expanded until a set of points is found that uniquely determines the plane.

4.4.2.3 Divergence-model approximation

The plane obtained using (4.15) is assumed to be an approximation of the tangent plane of the true task function in the vicinity of the point in question. This approximation is expected to diverge substantially as we move away from the fitted neighborhood. Therefore, we then estimate how quickly this divergence occurs by computing the absolute error $e^{(j)}$ between the predicted task score (using plane approximation at $x^{(i)}$) and the actual measured score for every point $x^{(j)}$ in an annular-box-neighborhood $\mathcal{M}^{(i)}$:

$$\mathcal{M}^{(i)}(\beta) = \{x^{(j)} : x^{(j)} \in \mathcal{X} / \mathcal{N}^{(i)} \wedge ||x^{(i)} - x^{(j)}||_\infty \leq \beta\} \tag{4.16}$$

where, $\beta > \delta$ is the size of the neighborhood.

$$e^{(j)} = |y^{(j)} - \mathcal{F}^{(i)}(x^{(j)})| \; \forall \; x^{(j)} \in \mathcal{M}^{(i)} \tag{4.17}$$

where, $\mathcal{F}^{(i)}$ is the affine hyperplane corresponding to $x^{(i)}$.

For each $x^{(i)}$, we then construct an error estimate function $\mathcal{E}^{(i)}(\Delta x)$ that estimates the upper bound on these absolute error values. Our formulation uses a quadratic function with different weights $\omega_j^{(i)}$ for each parameter axis and whose minimum lies at $x^{(i)}$ where the plane is fit:

$$\mathcal{E}^{(i)}(\Delta x) = \sum_{j=1}^{n} \omega_j^{(i)} \Delta x_j^2 \tag{4.18}$$

where, $\Delta x = x - x^{(i)}$. The weights $\omega_j^{(i)}$ are found by solving the following optimization problem:

$$\text{Minimize} \quad \sum_{j=1}^{n} \left(\omega_j^{(i)}\right)^2 \tag{4.19}$$

$$\text{subject to} \quad \mathcal{E}^{(i)}(\Delta x^{(j)}) \geq e_j \forall x^{(j)} \in \mathcal{M}^{(i)} \tag{4.20}$$

A quadratic programming solver was used for this purpose.

At the end of the model generation phase, we have $\mathcal{N} = \{\mathcal{N}^{(i)} : i = 1, 2, \ldots, m\}$, $\mathcal{F} = \{\mathcal{F}^{(i)} : i = 1, 2, \ldots, m\}$ and $\mathcal{E} = \{\mathcal{E}^{(i)} : i = 1, 2, \ldots, m\}$ representing the sets of m adaptive neighborhoods, m hyperplanes, and m error estimate functions, respectively, corresponding to each sample point in $\mathcal{S}$.

4.4.3 Exploitation-Driven Model Updating

Given a desired task score y_d, the sets $\mathcal{S}$, $\mathcal{F}$, and $\mathcal{E}$ are used to find a new point in the parameter space. As we want to search the parameter space quite conservatively, we would like to query a new point that will provide the smallest uncertainty in task score with respect to a local error estimate function. This is performed by selecting an existing sample point in $\mathcal{S}$ as a base point and by selecting only a single parameter for modification at that base point, which precludes the need to measure distances involving changes in multiple parameters and minimizes the possibility of error arising from unknown cross-effects between the parameters.

These two selections are made by conducting a search at each base point $x^{(i)}$ in the following way. For each parameter $x_j^{(i)}$, the desired corrective movement $\Delta x_j^{(i)}$ is calculated by finding a point in the direction parallel to that parameter axis whose task score based on the plane approximation is equal to the desired amount y_d. The parameter change is saturated if the corresponding error estimate function rises above a given threshold e_{max} before reaching the new point. Accordingly, the parameter change is given by

$$\Delta x_j^{(i)} = \operatorname{sgn}\left(\frac{y_d - y^{(i)}}{a_j^{(i)}}\right) \min\left(\left|\frac{y_d - y^{(i)}}{a_j^{(i)}}\right|, \sqrt{\frac{e_{max}}{\omega_j^{(i)}}}\right) \tag{4.21}$$

$$\forall\ j = 1, 2, \ldots, n.$$

The saturation limit on parameter change used in (4.21) prevents the system from testing points that have the potential for large error, possibly resulting in trials outside the proper operating range which would give no new information.

Note that the search in the parameter space is deliberately restricted to individual parameter directions. This results in generation of new sets of points called *line-sets*, where all points in a line-set $\mathcal{L}_j^{(i)}$ lie on a line parallel to single parameter axis j, $j = 1, 2, \ldots, n$:

$$\mathcal{L}_j^{(i)} = \{x^{(k)} \in \mathcal{X} : |x_\ell^{(i)} - x_\ell^{(k)}| \neq 0 \text{ only for } \ell = j\} \tag{4.22}$$

Therefore, whenever such a line-set is available for a base point $x^{(i)}$, the algorithm makes use of a line approximation over the points in the line-set, instead of using the planar approximation in (4.15), to compute the parameter change at that point. That is, for each parameter j where $|\mathcal{L}_j^{(i)}| \neq 0$, the algorithm computes $b_j^{(i)}$ as the slope of the best fit line through the points in $\{x^{(i)}, \mathcal{L}_j^{(i)}\}$. Accordingly, $b_j^{(i)}$ replaces $a_j^{(i)}$ in (4.21) during the computation of $\Delta x_j^{(i)}$.

Now, the error estimate function is computed for $\Delta x_j^{(i)}$ for all $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$ and the parameter change with the least error estimate is found as below:

$$\Delta x_{j^*}^{(i^*)} = \min_i \left(\arg\min_j \mathcal{E}^{(i)}(\Delta x_j^{(i)}) \right) \tag{4.23}$$

Therefore, this optimal amount of change will depend both on its magnitude, which is a function of the model coefficients $a_j^{(i)}$ or $b_j^{(i)}$, and how quickly the error function rises, which is a function of the quadratic surface weight $\omega_j^{(i)}$.

Now the new test point $\hat{x}$ is determined as follows:

$$[i^*, j^*] = \arg \Delta x_{j^*}^{(i^*)} \tag{4.24}$$

$$\hat{x}_j = \begin{cases} x_j^{(i^*)} + \Delta x_j^{(i^*)} \text{ if } j = j^* \\ x_j^{(i^*)}, \text{ otherwise} \end{cases} \tag{4.25}$$

The new point $\hat{x}$ is then sent to the trajectory generation module, which then provides the robot with a new trial. The robot performs the trial and the new task score $\hat{y}$ is recorded. Assuming the trial execution still results in failure, the new sample $(\hat{x}, \hat{y})$ is appended to $\mathcal{S}$ and the process is repeated.

4.5 Results

4.5.1 Synthetic Nonlinear Task Function

Our primary results are shown using a synthetic task of learning the behavior of a directly specified nonlinear function through sampling of the parameter space. An N-dimensional variation of the Schwefel function, whose landscape presents com-

Algorithm 3 Model generation and movement selection

Input: $\mathcal{S} = \{(x^{(i)}, y^{(i)}) : x^{(i)} \in \mathcal{X}, y^{(i)} \in \mathcal{Y}, i = 1, 2, \ldots, m\}$,
  target score y_d, success tolerance ϵ, time out t_{max},
  number of new sample points after each time out n_c
while ($\nexists\ y^{(i)} \in \mathcal{Y}$ s.t. $|y_d - y^{(i)}| < \epsilon$) **do**
  $t \leftarrow 0$;
  while ($t \leq t_{max}$) **do**
    for $i = 1 : m$ **do**
      $\left[\mathcal{N}^{(i)}, k\right] \leftarrow$ AdaptiveNeighborhoodSelection$(i, \mathcal{S})$;
      $A^{(i)} \leftarrow$ FitHyperplane $\left(\{(x^{(j)}, y^{(j)}) : x^{(j)} \in \mathcal{N}^{(i)}(k)\}\right)$
        using (4.15)
      $\mathcal{M}^{(i)}(\beta) \leftarrow \{x^{(j)} : x^{(j)} \in \mathcal{X} / \mathcal{N}^{(i)} \wedge ||x^{(i)} - x^{(j)}||_\infty \leq \beta\}$
      for $j = 1 : |\mathcal{M}^{(i)}|$ **do**
        $e^{(j)} \leftarrow |y^{(j)} - \mathcal{F}^{(i)}(x^{(j)})|$;
      end for
      $\omega^{(i)} \leftarrow$ FindWeights $\left(\mathcal{E}^{(i)}(\Delta x), f(\omega^{(i)}), \{e^{(j)} \in \mathcal{M}^{(i)}\}\right)$;
        using (4.18)−(4.20)
    end for
    for $i = 1 : m$ **do**
      for $j = 1 : n$ **do**
        $\mathcal{L}_j^{(i)} \leftarrow \{x^{(k)} \in \mathcal{X} : |x_\ell^{(i)} - x_\ell^{(k)}| \neq 0$ only for $\ell = j\}$;
        $c \leftarrow a_j^{(i)}$;
        if $|\mathcal{L}_j^{(i)}| \neq 0$ **then**
          $c \leftarrow FitLine\left(\{x^{(i)}, \mathcal{L}_j^{(i)}\}\right)$;
        end if
        $\Delta x_j^{(i)} \leftarrow \text{sgn}\left(\frac{y_d - y^{(i)}}{c}\right) \min\left(\left|\frac{y_d - y^{(i)}}{c}\right|, \sqrt{\frac{e_{max}}{\omega_j^{(i)}}}\right)$;
      end for
    end for
    $\Delta x_{j^*}^{(i^*)} \leftarrow \min_i \left(\arg\min_j \mathcal{E}^{(i)}(\Delta x_j^{(i)})\right)$;
    $[i^*, j^*] \leftarrow \arg\ \Delta x_{j^*}^{(i^*)}$
    $\hat{x}_j \leftarrow \begin{cases} x_j^{(i^*)} + \Delta x_j^{(i^*)} \text{ if } j = j^* \\ x_j^{(i^*)}, \text{ otherwise} \end{cases}$
    $\hat{y} \leftarrow$ Evaluate$(\hat{x})$;
    $\mathcal{S} \leftarrow$ Append$(\hat{x}, \hat{y})$;
    if $|y_d - \hat{y}| < \epsilon$ **then**
      success $\leftarrow 1$; **break**;
    end if
    $t \leftarrow t + 1$;
  end while
end while

Algorithm 4 Adaptive neighborhood selection

Input: Sample set $\mathcal{S}$, index of base point i;
$k \leftarrow n+2$;
$\mathcal{N}^{(i)}(k) \leftarrow$ GenerateNeighborhoodSet(k); using(4.13) & (4.14)
$\eta \leftarrow 0$;
for $j = 1 : k$ **do**
    $A \leftarrow$ FitHyperplane$(\mathcal{N}^{(i)}(k)/x^{(j)})$;
    $\eta \leftarrow \frac{\eta + \text{FindLinearFitError}(A, x^{(j)})}{k}$;
end for
$k^* \leftarrow k$;
$\eta_{prev} \leftarrow \eta$;
while (true) **do**
    $k \leftarrow k+1$;
    $\mathcal{N}^{(i)}(k) \leftarrow$ GenerateNeighborhoodSet(k); using(4.13) & (4.14)
    $\eta \leftarrow 0$;
    for $j = 1 : k$ **do**
        $A \leftarrow$ FitHyperplane$(\mathcal{N}^{(i)}(k)/x^{(j)})$;
        $\eta \leftarrow \frac{\eta + \text{FindLinearFitError}(A, x^{(j)})}{k}$;
    end for
    if $\eta_{prev} < \eta$ **then**
        $k^* \leftarrow k-1$; **break**;
    end if
    $\eta_{prev} \leftarrow \eta$;
end whilereturn $\left[\mathcal{N}^{(i)}(k^*), k^*\right]$

plexities including high nonlinearity and multiple local extrema was chosen for the purpose.

$$Schwefel : z_s(\mathbf{x}) = \frac{1}{2}\sum_{i}^{N} x_i \sin\sqrt{|cx_i|} \tag{4.26}$$

Figure 4.6 shows the landscape of a two-dimensional cross-section of the Schwefel test function, which indicates the complexity of the function and the challenge in learning its behavior. Our results were obtained with the five-dimensional version of the function, with each parameter in the range -1 to 1. For further compari-

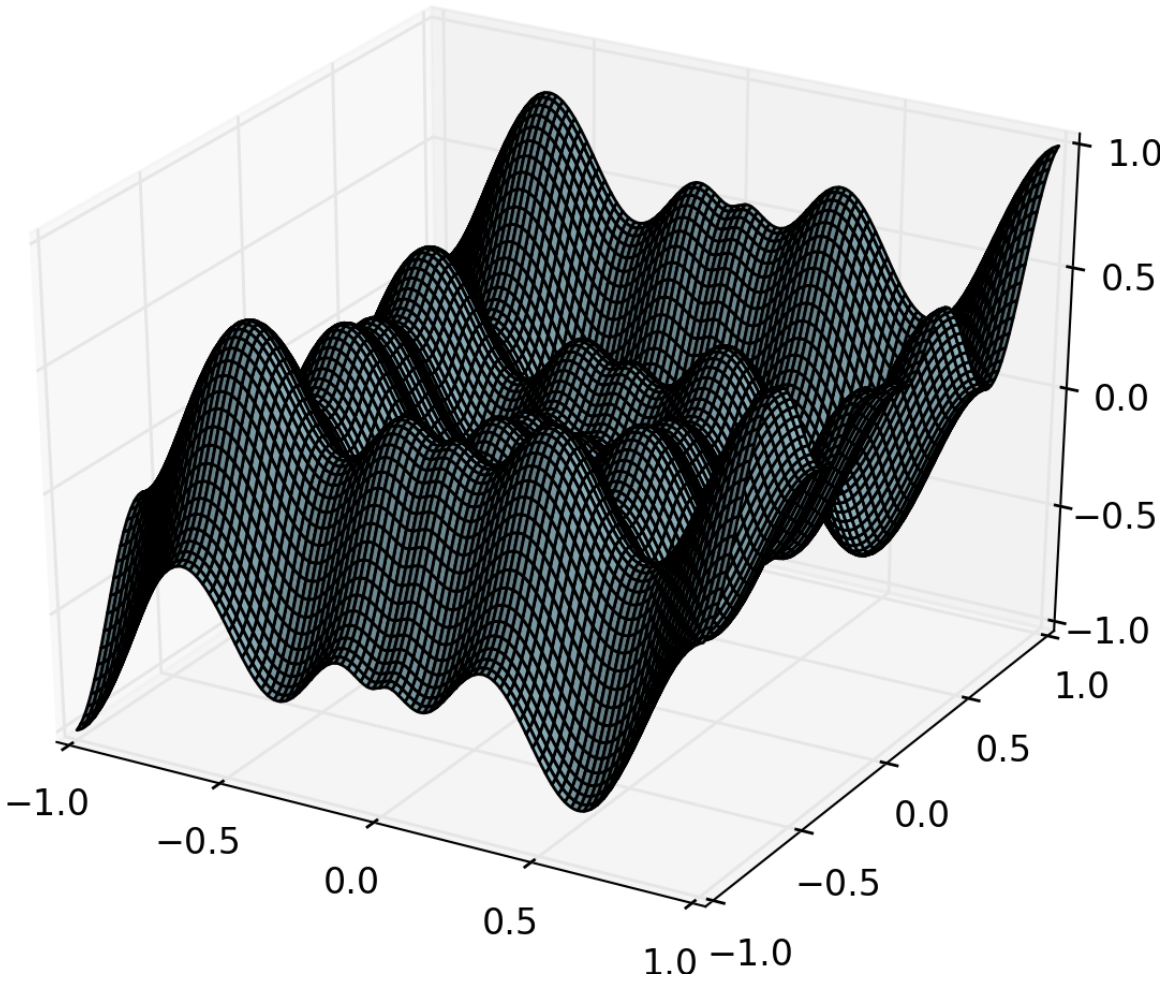

Figure 4.6: A cross-section of the Schwefel test function (c=200). Two parameters were varied from -1 to 1 and three were fixed at zero.

son, we additionally obtained results on a five-dimensional parabola with the same parameter domain.

4.5.1.1 Linear models exploitation performance

Figure 4.7 shows the performance as a function of the initial random data set size. As we expect to see, there is indication of asymptotic behavior where the algorithm is able to find a solution in a single iteration with sufficiently dense sampling of the parameter space. However, it is notable that the majority of the decrease in estimated number of iterations occurs at small data sizes (around 100), indicating good performance is attainable with only minor cost-intensive initial sampling.

Figure 4.8 shows the performance of our algorithm when tested with different values of task success tolerance. The algorithm is able to rapidly find solutions on both functions. The Schwefel function shows greater variability in the range of iterations needed, as would be expected given the much greater degree of nonlinearity exhibited. However, the maximum number of iterations needed is not significantly greater than the parabola result.

Finally, Fig. 4.9 shows the performance of the algorithm for a long-term learning scenario where all previous data points are retained . A set of 4500 targets was given to the algorithm with a sufficiently high tolerance that no existing points in the data set were already solutions. As we would expect, the performance of the learning algorithm approaches a single iteration when searching for a new target, indicating that the models formed by the training data approach the true function.

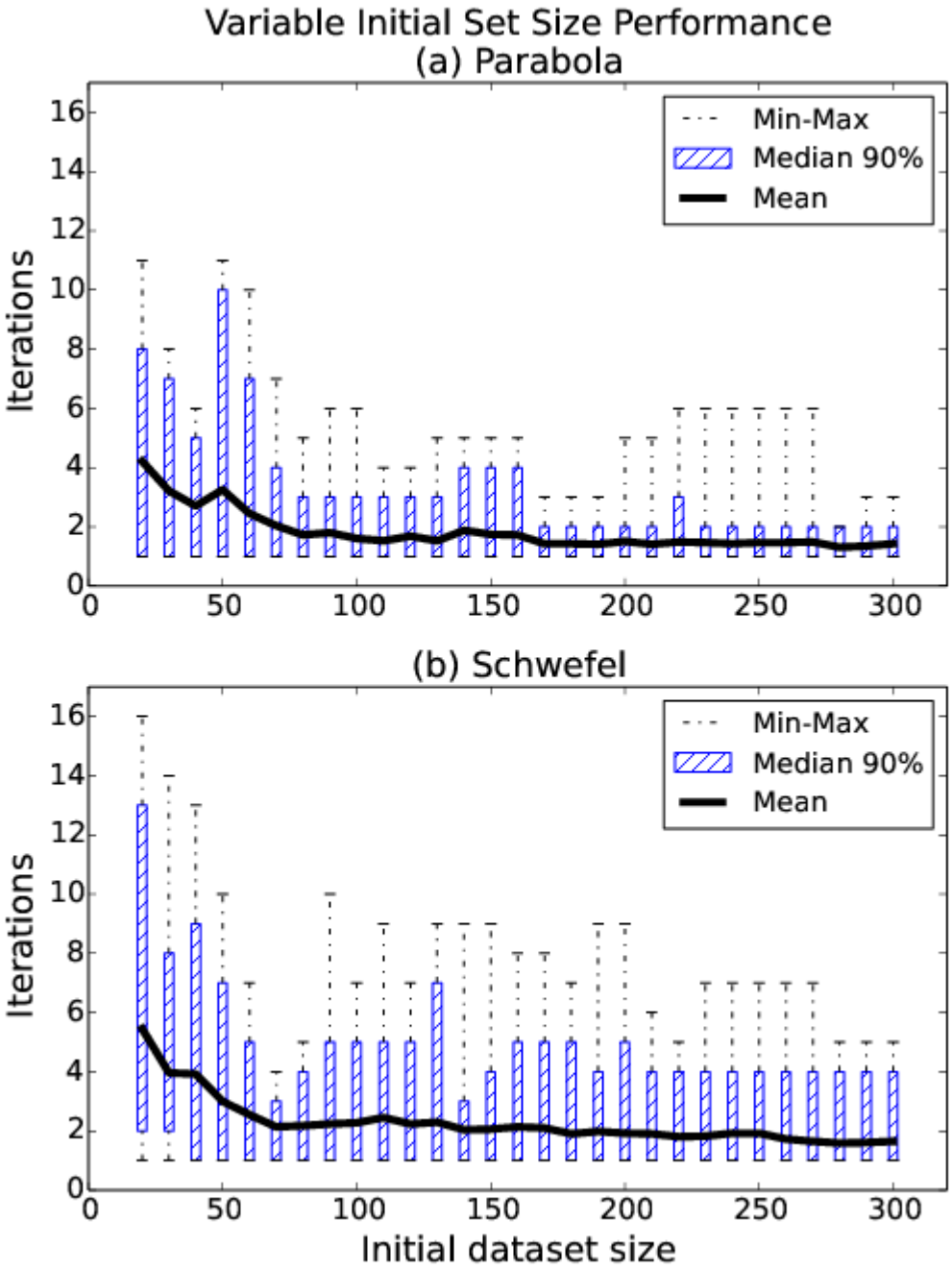

Figure 4.7: A comparison of the algorithm performance on both the (a) parabola and (b) Schwefel test functions, with varying sizes of the initial randomly sampled data set. All trials were performed with a tolerance of 0.005. Notice that, by design, the algorithm will always perform at least one iteration to minimize the error.

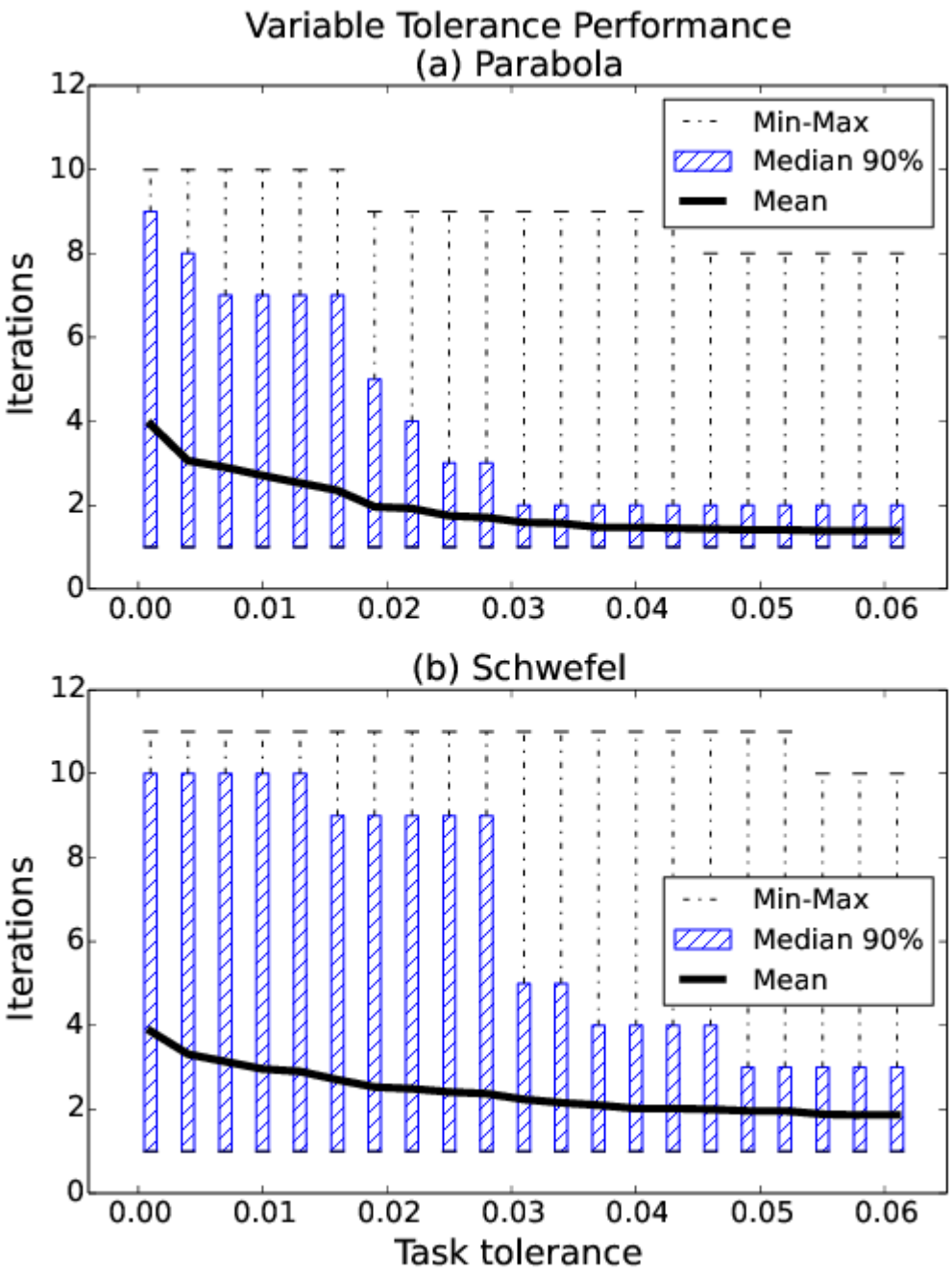

Figure 4.8: A similar comparison of the two test functions, but using different task tolerance values. All trials were performed with an initial data set of 50 sampled points.

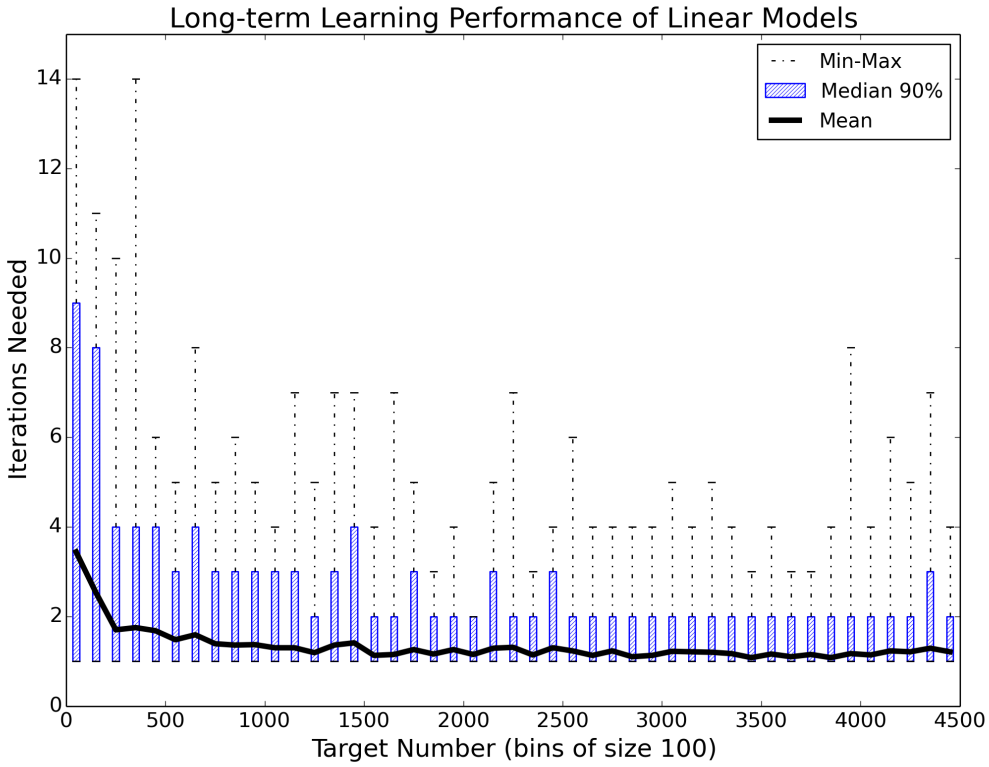

Figure 4.9: The algorithm performance for a set of 4500 randomly ordered targets where every trial is saved for future learning. The distribution of the number of iterations is shown for each bin of 100 consecutive targets to show the trend in performance.

4.5.1.2 Exploitation updates with a GPR model

As previously mentioned, it is possible to use many features of our exploitation updates approach in a regression-model-agnostic way. We therefore show the performance of our iterative strategy with an equivalent implemention on a Gaussian process regression model. A Gaussian process is trained on the same initial data set and can therefore make predictions both of the function value and the estimated error for new points. To implement the same exploitation-driven update strategy as described in section 4.4.3, we generate a set of parameter movements that are expected to acheive the desired target value and are based at the previously sampled data points. The only difference is that to find the move distances, an iterative search is required, as no invertible analytical model is available of the function behavior in the vicinity of the base point. All of the moves from the base points in all parameter directions are compared and the move with the smallest expected error is selected for execution. As before, if the target is not acheived within the specified tolerance, the true point is added to the library and the process is restarted.

To compare the two modeling algorithms, we gave each the same set of 50 data points and 100 targets to achieve. Unsuccessful trials were added to the set of points, but the data set was reset after each target was found. Figure 4.10 shows the number of these 100 targets achieved depending on how many iterations were run. Our approach with linear models has more targets found faster at 1-2 iterations but falls slightly behind for slightly longer runs of 3-6 iterations. Note that both approaches fail to achieve a few targets within 20 iterations (3 for our approach

and 13 using GPR). For our approach, this occurs because of a time-out in the updating algorithm, but we would expect the algorithm to eventually find solution points as more data samples were acquired. In the case of the GPR model, however, the missing targets are due to the search algorithm not finding any points with the expected target value, even when searching in all parameter directions from all known base points. In such cases, the updating strategy has no option but to exit and report that the target cannot be found. Finally, the computational cost of the our model was significantly lower as there is no need to search systematically for the estimate target. Due to this faster performance, initial performance benefit at low iterations, and the additional robustness for finding targets, we report the remaining results in this chapter for our model only. However, we still emphasize that the update strategy performed effectively with the GPR model and it may be more appropriate in other circumstances.

4.5.2 Algorithm Features Characterization

Here we present results demonstrating the value of the key algorithm features detailed in the previous section in terms of the overall performance. The three main features in our approach are (1) using single parameter adjustments, (2) using an upper-bounding quadratic error model, and (3) adaptively selecting the neighborhood size of each local model.

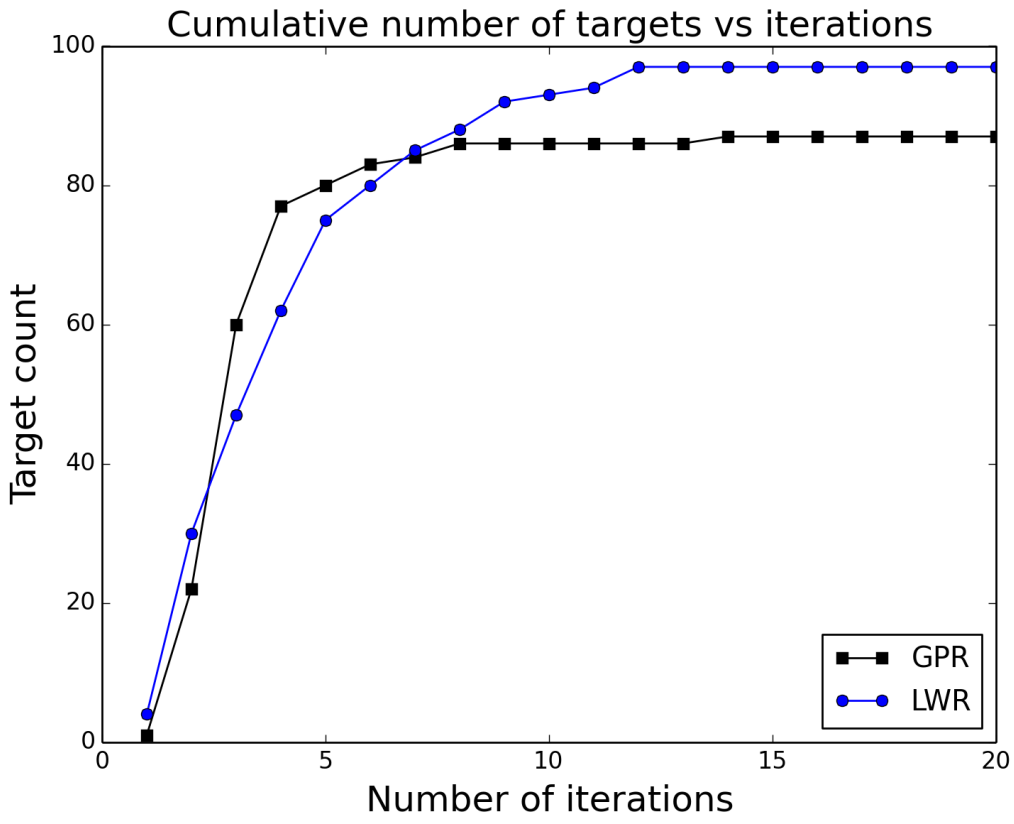

Figure 4.10: Comparison of the exploitation-update strategy using local linear models and a global GPR model in terms of the numbers of targets found by each iteration. The task tolerance was 10^{-3}.

4.5.2.1 Single parameter adjustments

One of the more novel features in our approach is that new test points are generating by only adjusting a single parameter from a previously tested point. While initally seeming overly restrictive, this has the benefit of generating multiple data points along a single line. We can then exploit this to do model fitting of that line directly, rather than only trying fit an N-dimensional plane to a neighborhood. This direct fit allows movements along the line to have much higher accuracy than along the plane in an arbitrary direction. Corresponding, the performance of the algorithm in terms of iterations needed to find desired task outputs is greatly improved.

We compare our approach with a simpler method where the movement from each known data point is parallel to the gradient of the plane fitted to the selected neighborhood. The adjustments are ranked by the expected error using the same error model and the movement distance is saturated if the expected error rises above a given threshold. So the only difference in the comparison is the direction of the adjustment and the possible use of the line model in the case of single parameter adjustments. Figures 4.11-4.12 show the results from the comparison. The single parameter adjustment strategy comes out substantially better, reducing the average number of iterations by over 50%.

4.5.2.2 Quadratic error model

For comparison with the quadratic error model, we used an algorithm that specifies a region around each local model as error-free and any prediction out-

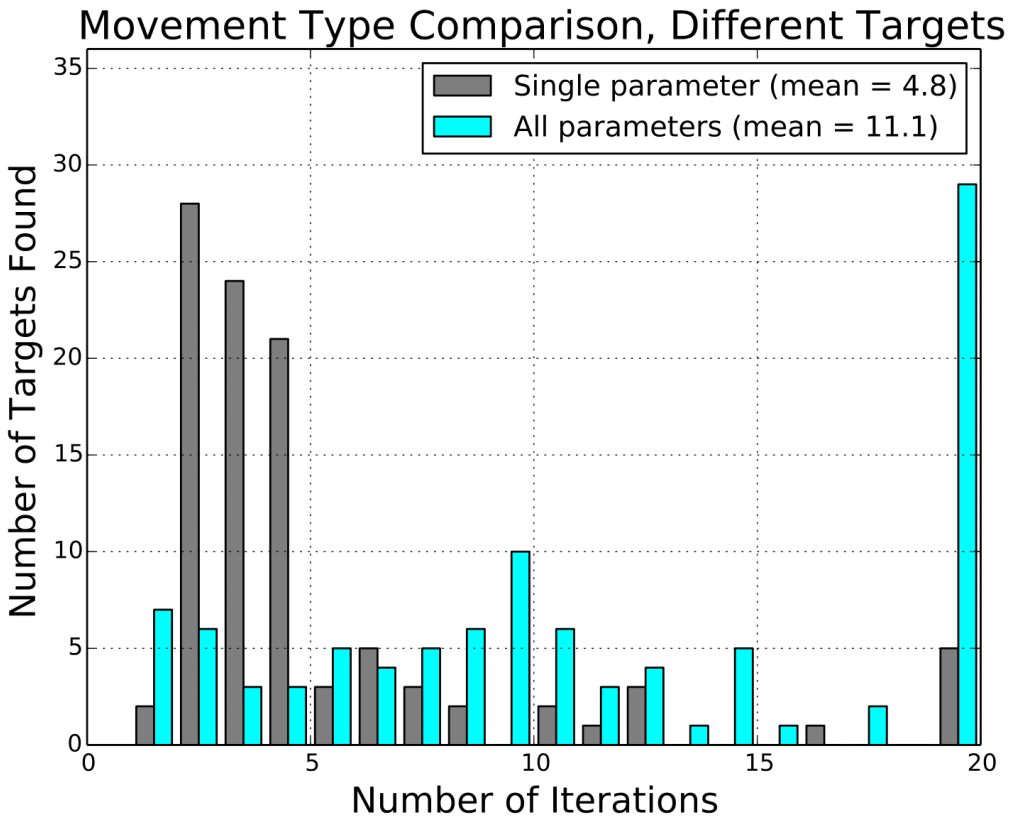

Figure 4.11: Performance of the learning algorithm using different parameter adjustment methods as a comparative histogram. The algorithm started with an initial data set of 50 random points and searched for 100 different targets.

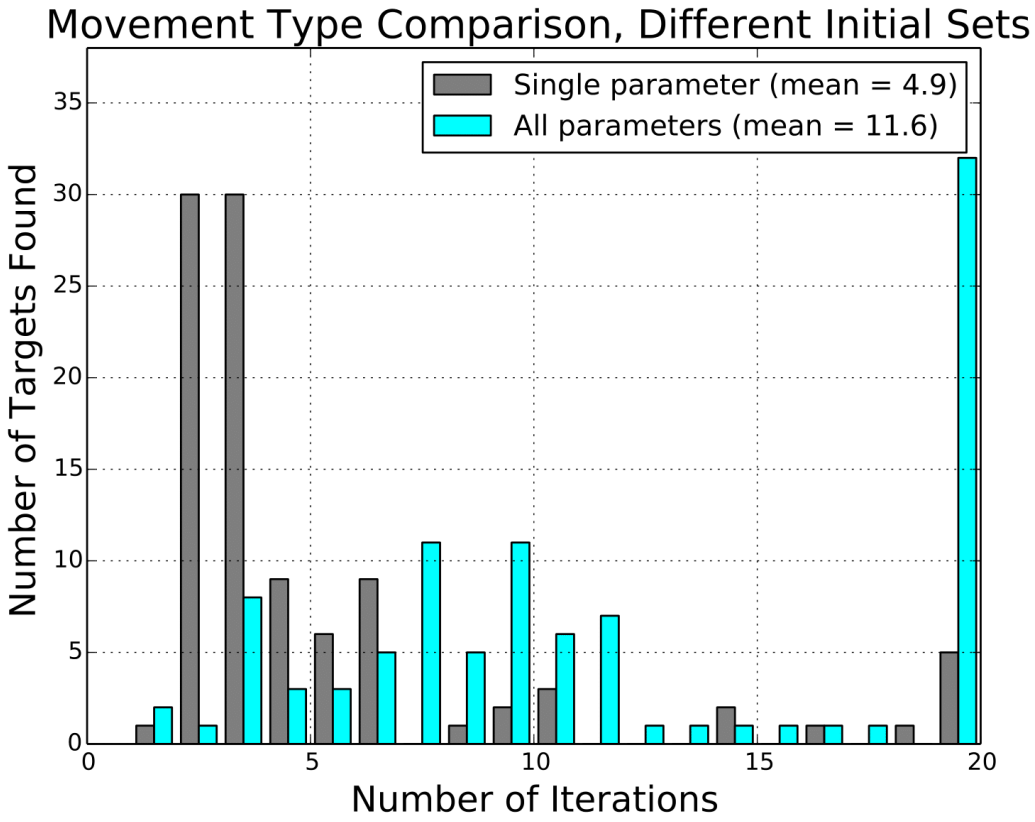

Figure 4.12: Learning algorithm performance with different parameter adjustment methods as in the previous figure. Here, the same target was found 100 times, but with a different random initial data set each time.

side of the region to have potentially infinite error, inspired by a trust-region approach [165]. The size of the region was determined by the distance to the furthest sample point whose measured error with respect to the plane fit to the local neighborhood remained below a threshold. If no neighboring point had an error below this threshold, the region radius was set to zero. The threshold can be set fairly arbitrary and we found that the model had good performance when it was set to the same value as e_{max}, described in section 4.4.3. When selecting the parameter adjustments from the different local models, the shortest move that remained within the error-free region was selected. The comparison results with the quadratic bounding error model are shown in Figs. 4.13-4.14, which shows better performance for the quadratic model on average. Interestingly, the error-free region model has better performance in the number of task instances solved after just two iterations, but then immediately drops off in performance while the quadratic model finds many other solutions in 3 and 4 iterations.

4.5.2.3 Adaptive neighborhood heuristic

The comparison for the adaptive neighborhood heuristic was done with two other much simpler heuristics. The N-closest heuristic uses the minimal number of neighboring points to fit the local plane, which is N for an N-dimensional task parameter space. If, as happens in the case of single parameter adjustments, the N closest points form a linearly-dependent vector space, the neighborhood was expanded incrementally until a unique plane could be fit to the points. For the fixed

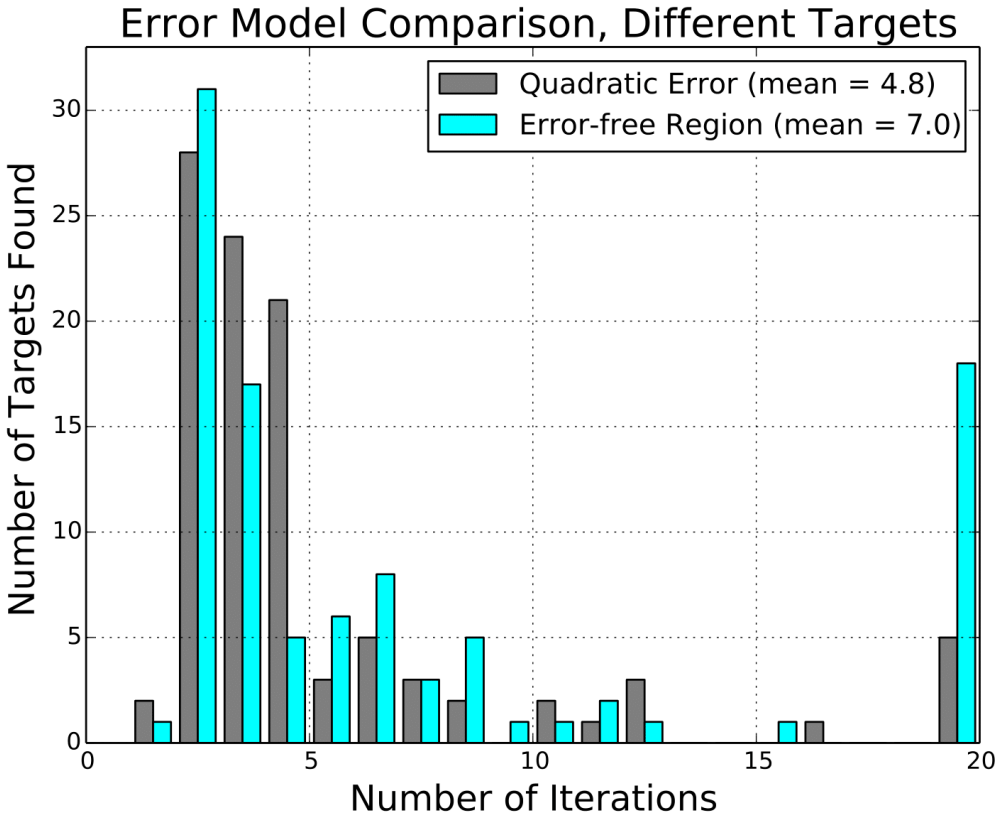

Figure 4.13: Performance of the learning algorithm using different error model algorithms as a comparative histogram. The algorithm started with an initial data set of 50 random points and searched for 100 different targets.

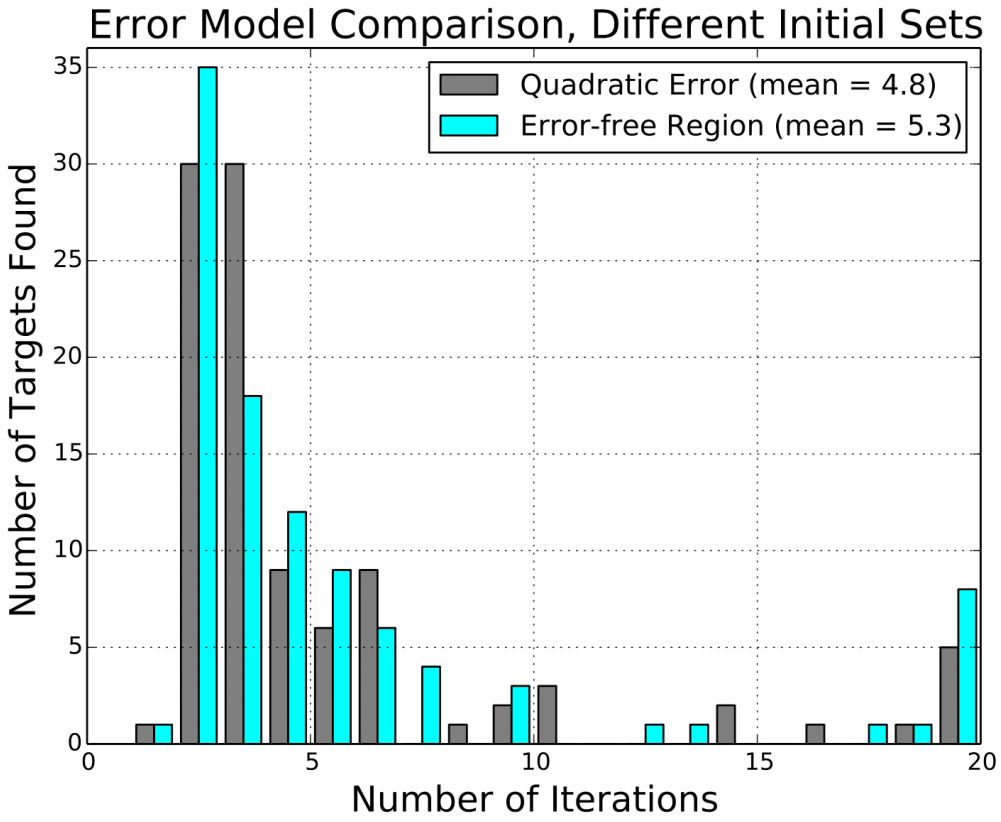

Figure 4.14: Learning algorithm performance with different error model algorithms as in the previous figure. Here, the same target was found 100 times, but with a different random initial data set each time.

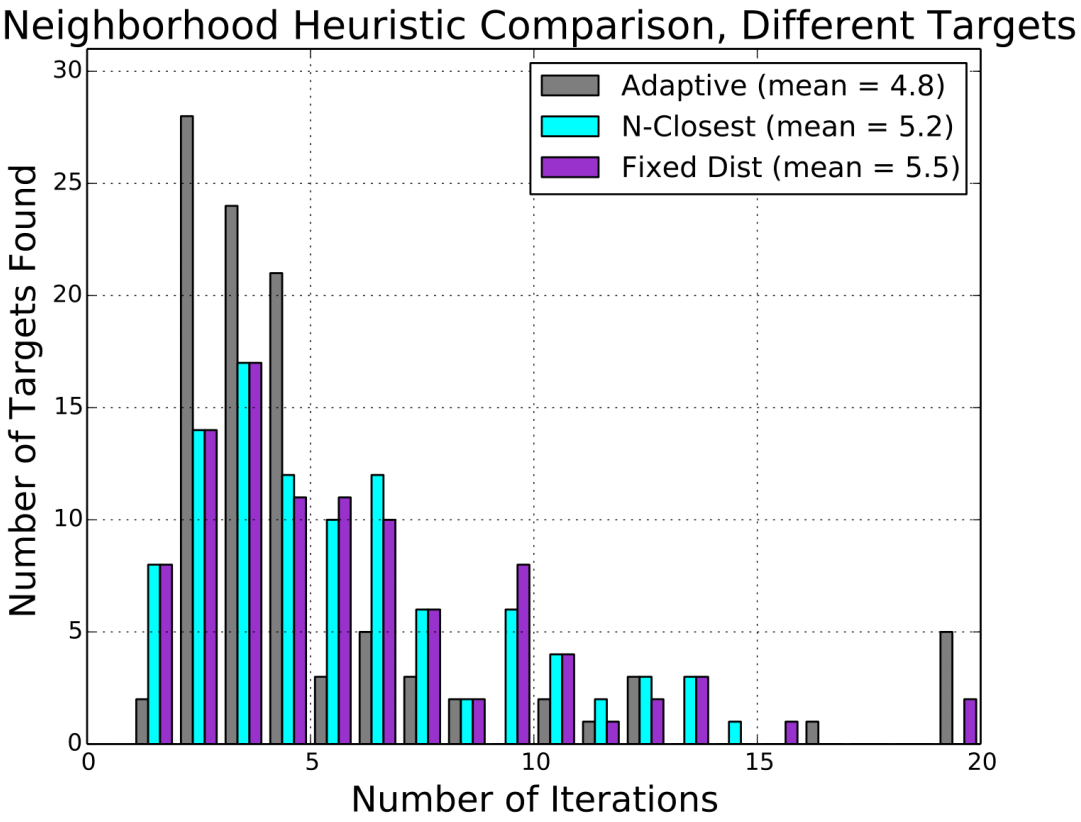

Figure 4.15: Performance of the learning algorithm using three different neighborhood heuristics as a comparative histogram. The algorithm started with an initial data set of 50 random points and searched for 100 different targets.

distance heuristic, all points within a specified distance were selected for the neighborhood. If there were insufficient points within the given distance, the results from the N-closest heuristic were used instead. Figures 4.15-4.16 show the results when compared with the adaptive neighborhood heuristic. On average, the adaptive heuristic has better performance, especially with the number of solutions found within 3 iterations.

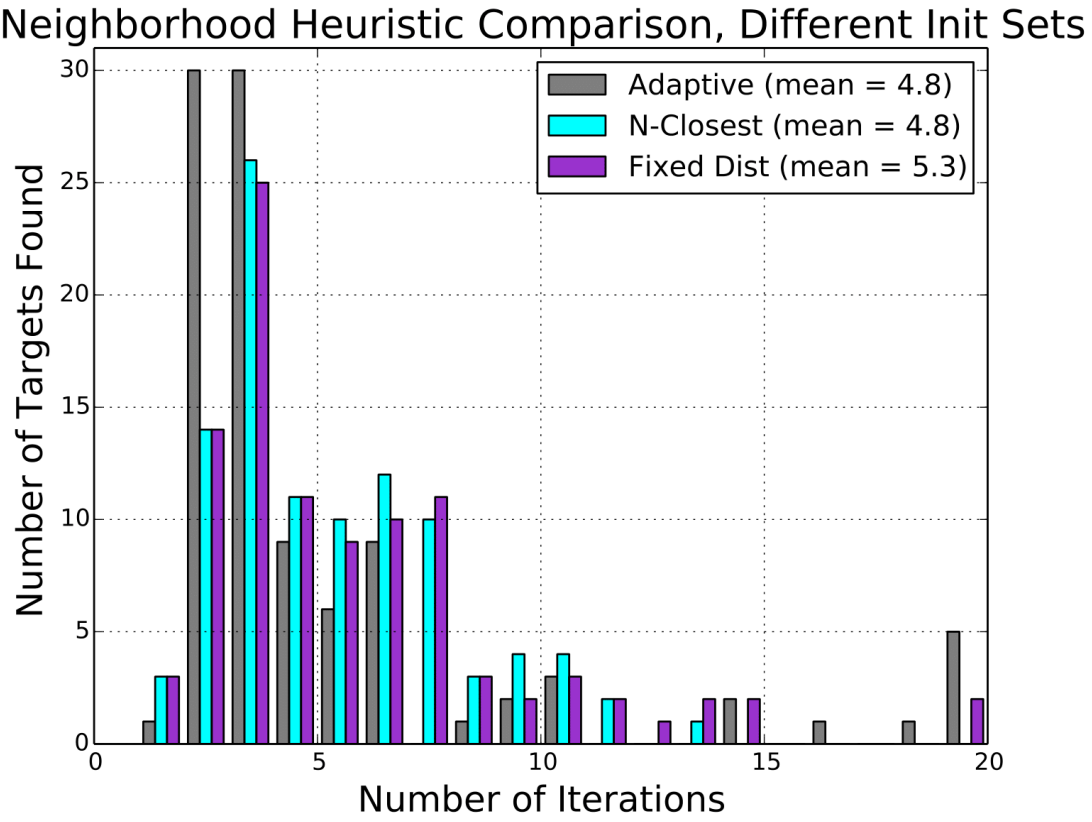

Figure 4.16: Learning algorithm performance with three neighborhood heuristics as in the previous figure. Here, the same target was found 100 times, but with a different random initial data set each time.

4.5.3 Robot Dynamic Pouring Experiments

To validate our approach on a physical robot, we used a dynamic pouring task, where a Baxter robot is tasked to pour a specific amount of liquid into a container which is placed on a rotating table. This task is intended to be representative of a manufacturing scenario where the robot may be asked to perform new tasks or task variations with limited time available for learning. Additionally, the task of pouring liquid into a moving container is highly amenable to autonomous learning as it is extremely difficult to model accurately without an existing data set.

We selected the tilt trajectory of the bottle held by the robot as the learning target with the complementary motions of the robot's joints found using standard planning algorithms. In particular, the tilt profile robot's end-effector action consists of tilting the bottle in one direction (forward tilt) for some duration, keeping the tilt steady for some time, and untilting the bottle (reverse tilt). Accordingly, the tilt profile was parameterized by five real-valued parameters, which were manually defined as relevant physical features of the pouring action: (1) forward tilt time t_f, (2) forward tilt rate $\dot{\theta}_f$, (3) intermediate pouring time t_s, (4) reverse tilt time t_r, and (5) reverse tilt rate $\dot{\theta}_r$. A value for each of these parameters defines a point in the five-dimensional parameter space in which the learning algorithm operates. A example plot of what the generated tilt trajectory looks like can be seen in Fig. 4.17.

The experimental setup used is shown in Fig. 4.1. The rotating table has several visual markers that the robot can use to estimate its position as well as

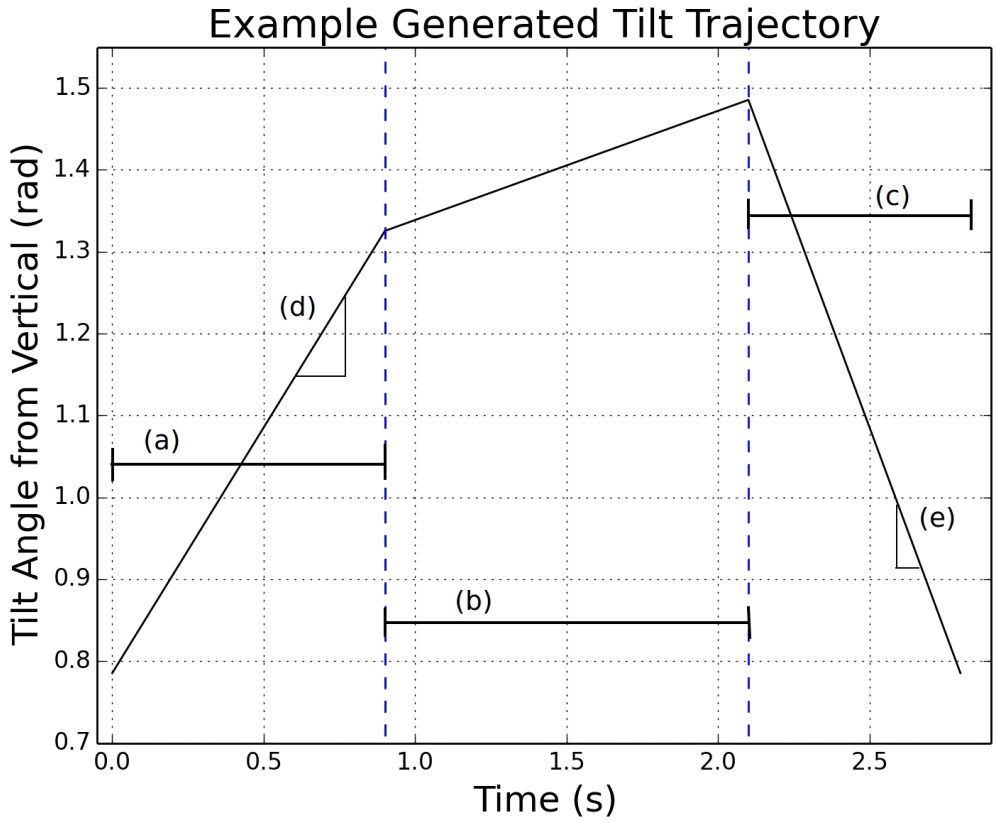

Figure 4.17: An example of a generated tilt trajectory which is controlled by five parameters: (a) the tilt-forward time, (b) the intermediate time between the two tilts, (c) the tilt-backward time, (d) the tilt-forward rate, and (e) the tilt-backward rate. To reduce the amount of overall rotation needed, the base tilt angle is not zero but 45 degrees.

Figure 4.18: An example execution of the pouring task for illustration. The four images show (a) the robot holding the bottle and waiting for the trigger to begin pouring, (b) the initial tilting phase where the liquid starts to begins to rush out, (c) the middle pouring phase where the flow is more laminar, and (d) the untilting phase where the flow is cut off by the rising bottle edge.

current angle. After measuring the table rotation speed, the planning algorithm takes the tilt profile to execute and generates a path for the arm which ensures the tip of the held bottle will remain above the target container during the pouring motion. An example execution of the task is shown in Fig. 4.18 (using the tilt-trajectory in Fig. 4.17), which shows the breakdown of the pouring motion into the three phases controlled by the learned parameters.

After each pour, the amount of fluid poured was measured manually using a scale with a precision of 2 grams. Initial exploratory tests showed that due to the various system errors, including measuring the table position and robot trajectory tracking, the amount poured for a single set of parameters has a variation of up to ± 15 grams from a mean poured volume. This indicates a small degree of stochastic behavior that can affect the number of iterations needed to find a pouring trajectory which falls within a strict tolerance of task success. For context, it should be noted that the variation in poured volumes when the task is performed by humans, even with practice, is much higher that the robot variation.

An initial library of 40 points was generated by evaluating randomly generated points in the parameter space. Three were removed where either the entire bottle of fluid was poured (450 grams) or no fluid was poured. With this initial data set, twelve targets were given uniformly from 100 to 400 grams. Figure 4.19 shows the performance on these targets for three different task success tolerance values. In keeping with our results from function approximation, the algorithm has faster convergence with less restrictive tolerance. Figure 4.20 shows a second experiment where the same targets were repeated but the initial data set used included the

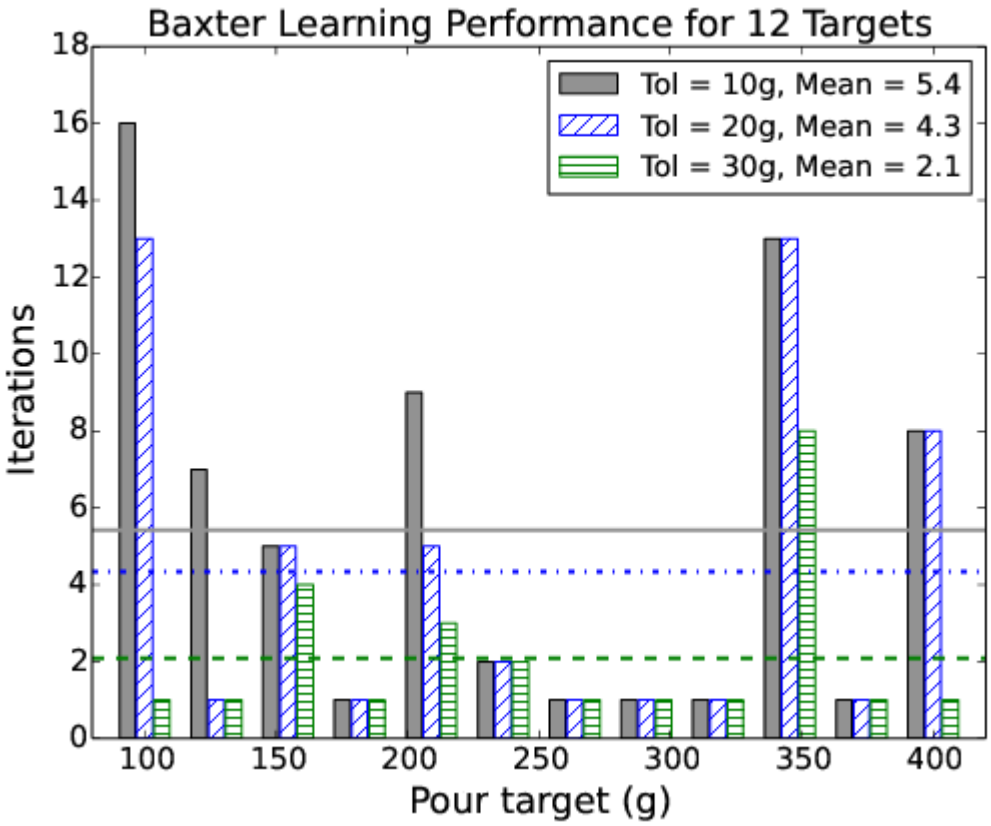

Figure 4.19: Baxter learning performance for a uniformly distributed set of targets. Each group of three bars is the number of iterations needed for a single target, with the gray, blue, and green bars corresponding to tolerances of 10, 20, and 30 grams, respectively. All trials were done with the initial set of 37 random points.

points tested in all previous targets. Again, paralleling the results from the previous section, there is a visible trend of a decreasing number of iterations needed as the initial library size grows. Also note that the all the means for the long-term learning in Fig. 4.20, using a 20 gram tolerance, are lower than the mean for the 20 gram tolerance case in Fig. 4.19.

4.6 Summary

This chapter presents an approach that allows a robot to generate task trajectories using a set of linear metamodels. These models were successfully learned from sparse initial exploratory experiments and enabled the system to learn to perform a complex task instance with very few attempts needed. Our approach made use of multiple features which were demonstrated to improve the overall performance of the algorithm, including single parameter adjustments, local quadratic error models, and adaptive neighborhood selection. Experimental results using both synthetic and real robot tasks revealed that the algorithm converges faster as more experiments were conducted, suggesting that the algorithm supports lifelong learning. This work also discussed how alternative well-known approaches were not ideal for solving this particular problem, which the presented approach handles quite well. In summary, the approach has been shown to be useful for motion planning problems where many similar task instances must be solved and in which model prediction by simulating the underlying physics involving the trajectory variables and task behavior is very difficult. Provided that the task space has many solutions available to find, the approach is capable of very quickly finding the solutions, even for highly nonlinear tasks with very precise tolerances required.

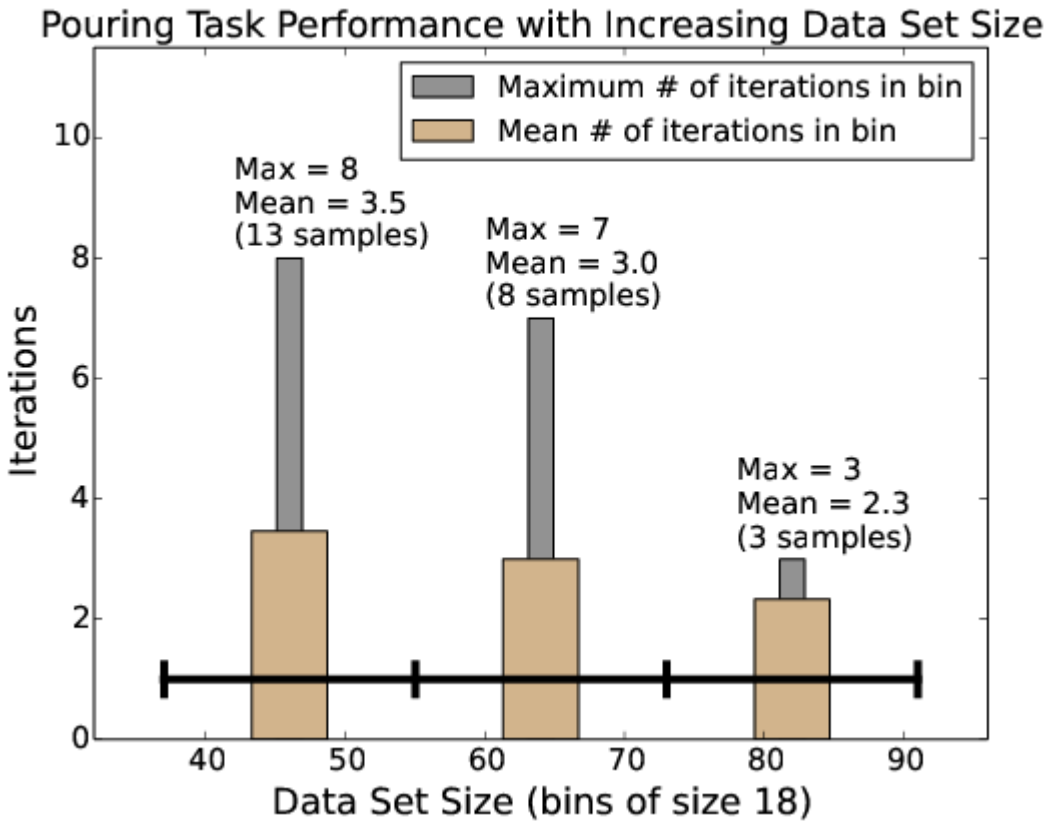

Figure 4.20: Baxter learning performance depending on initial size of the data set when starting each new target. For this data, the robot was given the same initial set of 37 samples, but as it progressed through the 12 targets, all trials were saved in the data set so later targets had more samples to learn from. The full experiment was then repeated starting from the initial set once again to obtain 24 total target samples. As there was substantial variation in the samples, the results were divided into three bins so that a rough mean could be estimated. The success tolerance was 20 grams for all targets.

Chapter 5: Task Cost Function Estimation from Demonstrations in Cleaning of Elastically Deformable Objects

5.1 Introduction

There are many manipulation tasks that robots are currently capable of performing but are difficult to program. One large source of difficulty arises for many tasks where humans have developed a natural and efficient understanding of how to complete the task, but are unable to articulate their strategy as a set of rules that can be transfered to the robot. This is evident in tasks involving the manipulation of deformable objects, and in particular, performing finishing actions such as cleaning or polishing on such objects. Cleaning involves some level of pressure on the part surface and so the resulting deformation of compliant parts must be taken into account during the task execution and planning. Through repeated experience, humans acquire a balanced method of handling such scenarios, where they choose grasps on the object that minimize the deformation while still finishing the task in a timely manner. However, it can be difficult to convey this balance to another system in a direct manner.

Learning from Demonstration (LfD) is an active field of research with the goal of teaching robots the strategy to perform new tasks by only observing how humans perform them. One particular area in the LfD field involves extracting a cost function that the demonstrator can be said to be implicitly using when performing the task. This has some analog in the neuroscience realm, where separate areas of the brain function at higher levels of the task than the pure motion planning level [166]. By learning the task cost function used by the human, the robot is able to plan its behavior for a large set of possible task variations, enabling it to have much general behavior than possible when only taught specific task instances.

This work presents an approach for estimating the parameters of a cost function being used by a human when performing a cleaning task. The core contribution is a method that allows for estimation without assuming that the demonstrator has total familiarity with the stiffness properties of the part being cleaned. This allows the function to be learned while the demonstrator is simultaneously improving their knowledge of the part. A robot system can then use the estimated cost function parameters in cleaning of other parts and maintain a balance between speed and deformation in the same style as humans.

5.2 Approach

The primary goal of our approach is to uncover the implicit cost function the human demonstrator uses when performing and learning the finishing task. This cost function is assumed to be a linear function of the task features, of the form

$$C(\mathbf{x}) = \mathbf{w}^T \mathbf{x}. \tag{5.1}$$

By estimating the value of the weight vector $\mathbf{w}$ used by the demonstrator, the robot can learn to perform new task variations with similar behavior as humans.

5.2.1 Cost Function Structure

We assume that the cost function influencing the demonstrator behavior has two main factors: the task time/speed and the amount of deformation imposed on the part. These two components are directly competing, as it is generally possible to perform a finishing task with little deformation (using many grasping locations) but with the penalty of more task time. As such, we define the function to use only a single weighting parameter controlling the convex combination of the two factors:

$$C_w(\mathbf{x}) = C_w(x_1, x_2) = wx_1 + (1 - w)x_2. \tag{5.2}$$

This is functionally equivalent to having two separate weights (since only relative cost values matter) but allows us to fix the unknown parameter in the normalized range $[0, 1]$ rather than be any real number, which simplifies the estimation process.

Many different features may be possible to use in the feature vector to represent the competing factors in the task execution. For this work, we take the time/speed factor to be the number of grasps used during the cleaning attempt and the average flexibility experienced over the stained regions given the grasping location at the time of cleaning. The number of grasps is used rather than the time directly as it does not include the time needed to operate the grasping tool, which contains substantial noise. Additionally, the time required to clean a stained region is roughly independent of the level of flexibility present, provided that it remains within a reasonable range.

Suppose a human demonstrator executes a cleaning trial $\phi = \{(g_1, s_1), ..., (g_N, s_N)\}$, where g_i is the grasping location used while cleaning stain s_i. Then we define N_g as $|\{g_1, ..., g_N\}|$, the number of unique grasping points. We define the flexibility factor as

$$F = \frac{1}{N} \sum_{i=1}^{N} \mathcal{F}(g_i, s_i), \tag{5.3}$$

where $\mathcal{F}(g, s)$ is a known function for a particular part that provides the flexibility between particular grasp and stain points in units of length/force. The cost of task trial can then be computed from Eq. 5.2 using the feature vector $\mathbf{x} = [N_g, F]$.

5.2.2 Probabilistic Plan Generation

Computing the likelihood of a particular weight parameter value can be achieved by computing the probability of the observed data assuming it was generated with that parameter value. Here we detail this probability calculation. We assume that the demonstrator generates the cleaning trial with imperfect knowledge of the part model which improves over the set of trials. We represent this imperfect knowledge with a probability distribution for the flexibility function $\mathcal{F}$. The flexibility of the part between a particular grasp point and tool contact point can range from 0 to infinity, and so we use a log-normal distribution over the flexibility to assign probability density over this range. This is achieved simply by scaling the known true flexibility of the part by a random variable taken from a log-normal distribution

The next step is being able to compute the probability that a particular plan is generated by the demonstrator given a cost weight value. The plan generation necessarily requires some level of solving an optimization problem and so we cannot specify a closed-form transformation from the cost probability to the plan probability. Rather, we compute the plan probability using a Monte Carlo method with a stochastic optimizer.

Given a variance parameter value to control the distribution of the values of $\mathcal{F}$, many samples are generated of different part models with varying level of flexibility and grasping plans are generated by optimizing for the assumed cost function. An off-the-shelf stochastic optimizer [167] is used to ensure some additional randomness in the resulting generated plans. The resulting distribution of generated plans is then

used to estimate a discrete probability mass function describing the probability of the number of grasps used simply by counting the number of generated plans with that number of grasps. The resulting probability of the observed plan is reported as the probability mass fraction matching that number of grasps. Note that the computation result can be identically zero. For example, if w is taken to be close to 1, the set of plans generated during optimization is very unlikely to contain any sample with a large number of grasps, due to the strong influence of the term penalizing the total number of grasps. If the observed plan has several grasps, it will be assigned a probability of zero as all the probability mass will be placed on plans with a small number of grasps. Another important note is that this strategy will fail if the optimizer finds a true minimum solution each time, resulting in the set of generated plans being a single point. This can be addressed by requiring some minimum bounds of uncertainty are used on the flexibility function. The formal description of the steps involved in the probability computation are described in Algorithm 5.

5.2.3 Parameter Estimation

Finally, we can use this probability calculation to estimate the cost function used by the demonstrator in the cleaning task, also described in Algorithm 5. We assume that the demonstrator has converged on the value of weight parameter after the first cleaning task and is using that value as they perform the cleaning trials for the second task. We also assume that the demonstrator begins with a relatively

Algorithm 5 Probability and likelihood computations of observed plans

function ComputeProb(ϕ, $\mathcal{F}$, σ_F, w)
  for $i \leftarrow 1$ to N_{samp} **do**
    $\mathcal{F}' \leftarrow$ Sample($\mathcal{F}$, $log\mathcal{N}(\mu = 0, \sigma_F)$
    $\phi'_i \leftarrow$ Optimize($\mathcal{F}'$,w)
  end for
  for $j \leftarrow 1$ to N_g **do**
    $P(|\phi| = j) \leftarrow |\{\phi'_i s.t. |\phi'_i| = j\}|/N$
  end for
  return $P(|\phi| = m)$ with $m = |\phi|$
end function

function ComputeLikelihood(w, Φ, $\mathcal{F}$, Σ_F)
  for $i \leftarrow 1$ to N_{trials} **do**
    $P(\phi_i|w) \leftarrow$ ComputeProb(ϕ_i, $\mathcal{F}$, σ_i, w)
  end for
  return $\mathcal{L}(w) = \prod_{i=1}^{N_{trials}} P(\phi_i|w)$
end function

unknown estimate for the part flexibility function which converges to close to the true value by the final trial, though some variance is still used even in the final trial. A set of variance parameters $\{\sigma_i : 1 \leq i \leq N_t\}$ is used to specify the distributions of scale factors sampled for the modified flexibility function.

The choice of how to schedule the variance parameters over the range of trials is fairly arbitrary and future work might focus on developing a rigorous criteria for selecting optimal values. For this work, the schedule selected was a geometric series controlled by two freely-selected parameters: the variance during the final trial σ_f, and a scaling parameter σ_{sc}. Starting with the final trial, the variance parameter for each preceding trial is computed by scaling by σ_{sc}. As long as $\sigma_{sc} > 1$, the variance will be larger for earlier trials. This geometric sequence of variance parameters has

the advantage that the largest decrease in variance occurs after the first few trials, which corresponds to when humans learn most of the general part behavior.

Using the resulting distributions of $\mathcal{F}$ for each trial and given a value of w, we can compute the joint probability of the full set of plans Φ used by the demonstrator, which is simply the product of the probabilities for each trial.

$$P(\Phi|w) = \prod_{i=1}^{N_t} P(\phi_i|w) \tag{5.4}$$

We then use maximum likelihood estimation to compute the most likely value of w. By the definition of likelihood, $\mathcal{L}(w|\Phi) = P(\Phi|w)$, and so we can estimate w by varying it in the range of 0 to 1 and observing where the probability is maximized:

$$\hat{w} = argmax_{w \in [0,1]} \mathcal{L}(w|\Phi) \tag{5.5}$$

Typically, a closed-from expression can be found for $P(\Phi|w)$, allowing for the possibility of finding the maximum value by making use of an analytical derivative. However, since our calculation of the plan set probability involves a stochastic search, there is no closed-form available, and the maximum must be found through a black-box optimization process.

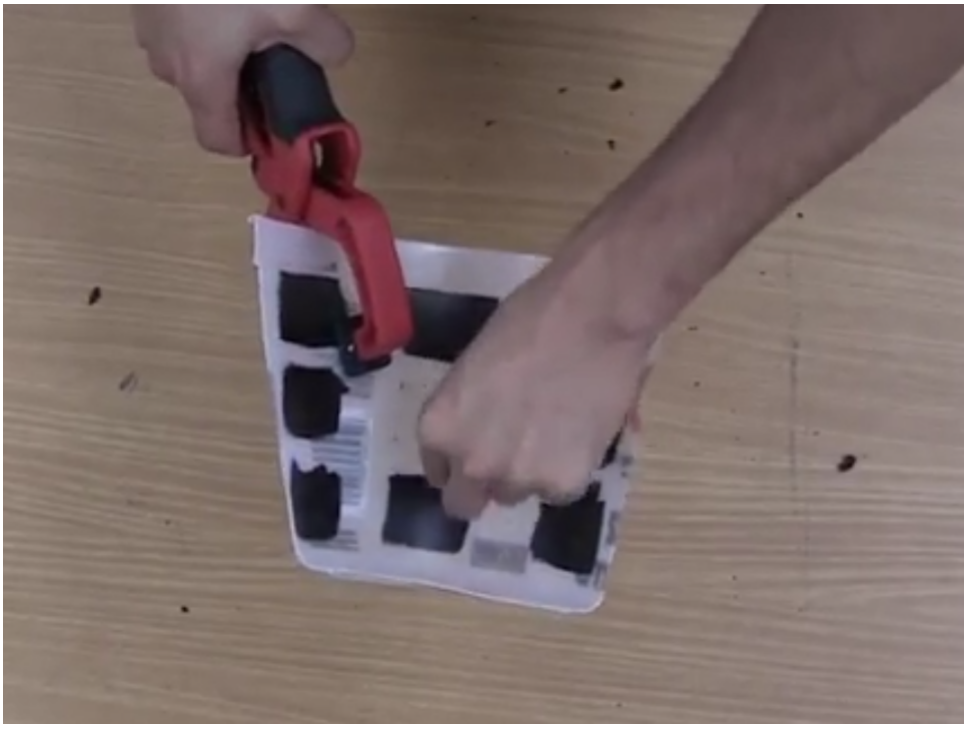

Figure 5.1: Cleaning trial with moderate deformation

5.3 Results

5.3.1 Cleaning Task Details

The cleaning task used for results consisted of two compliant plastic parts, both with similar geometries and similar material composition but of different size, which affected their overall flexibility. Ten stains were placed on each part in a uniform pattern and the demonstrator was asked to clean each stain sequentially while holding the part above a table. In order to keep the grasping data easily recordable and provide behavior reproducible by a robot, the subjects were required to use a clamping tool to hold the part during cleaning instead of their hands. As the subjects cleaned the stains, they were allowed to relocate the clamp anywhere

Figure 5.2: Cleaning trial with heavy deformation

and as many times as they desired. However, they were instructed that the goal was to finish the trials as quickly as possible while remaining within a comfortable range of deformation. Figures 5.1-5.2 show the overall experiment setup and example demonstrator behavior during two cleaning trials, one with relatively low deformation, and one with quite high deformation.

5.3.2 Collected Data

In total, 10 subjects were used for data collection, each performing 10 cleaning trials. From the video data of each trial, the total cleaning time, the order that the stains were cleaned, and where the demonstrator gripped the part was recorded. As the data was obtained through visual inspection of the video, the grasping positions

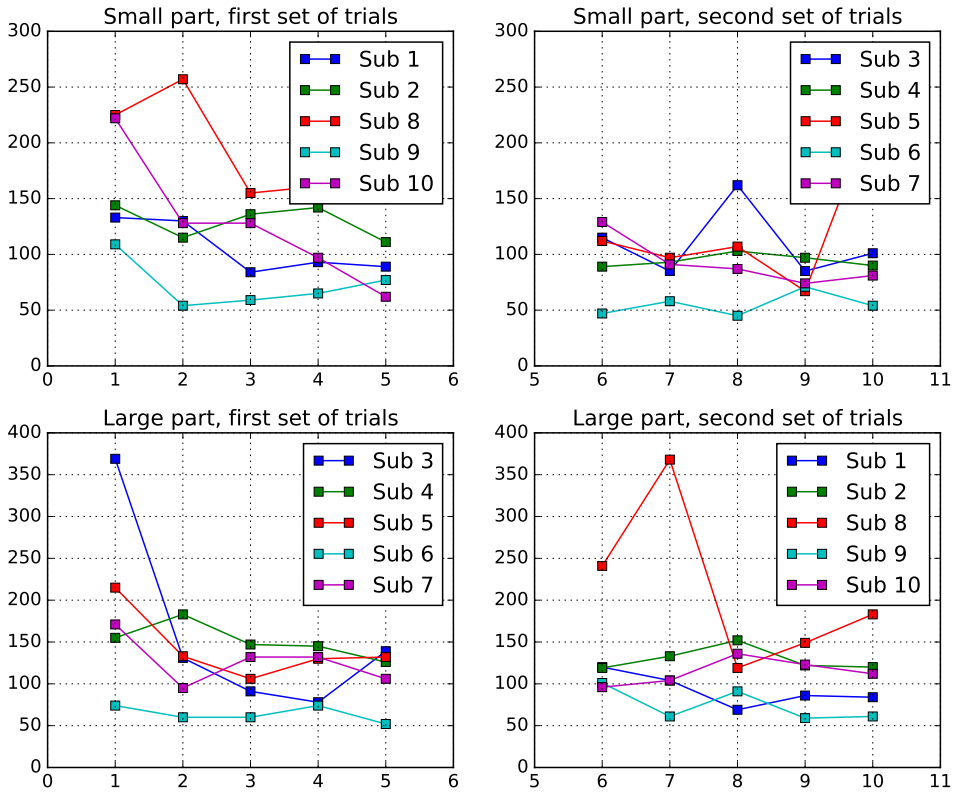

Figure 5.3: Cleaning trial times for the 10 subjects

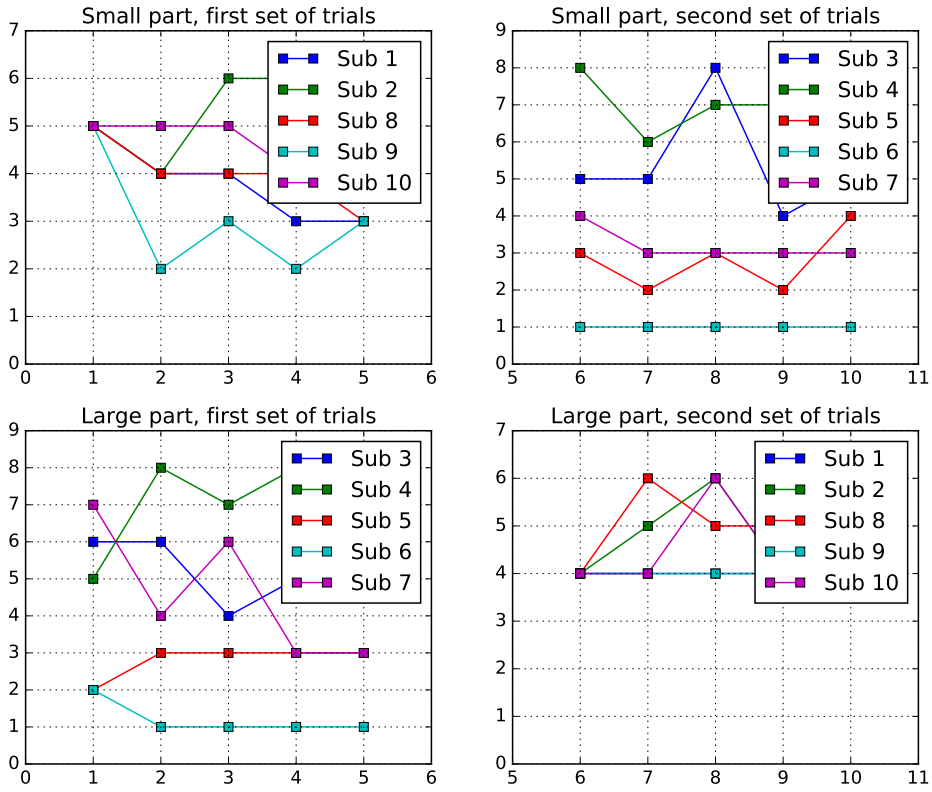

Figure 5.4: Number of grips used during cleaning trials for the 10 subjects

were recorded as discrete values from a finite set, with a fixed number of points prespecified on each edge of the part, including the distance from the edge where the clamping tool made contact. The overall data in summarized in Figs. 5.3-5.4, which show the cleaning times and number of grips used in the cleaning trials, respectively. A few trials were invalidated due to improperly created stains that were far more difficult to clean than the others and would interfere with the task dynamics learning, as it assumes a uniform difficultly for all demonstrations.

Parameter estimation was done on the second set of trials performed by the demonstrators, after they had already cleaned one part, and developed a feel for the appropriate balance between cleaning time and part deformation. An example results of the resulting likelihood computation over the set of trials for various parameter values can be seen in Fig. 5.5, where a clear maximum is calculated at $w = 0.2$. This indicates a preference for less part deformation at the expense of more grasping point. This preference is confirmed by observing in Fig. 5.4 that the Subject 4 did indeed use more grasping points than typical during demonstrations of cleaning the smaller part second.

5.4 Summary

The chapter presents an approach to estimate cost function parameters from human demonstrations of tasks involving manipulation of elastically deformable objects. The presented approach enables the robot to acquire knowledge about how to perform the task that a human has implicitly gained through a large amount of

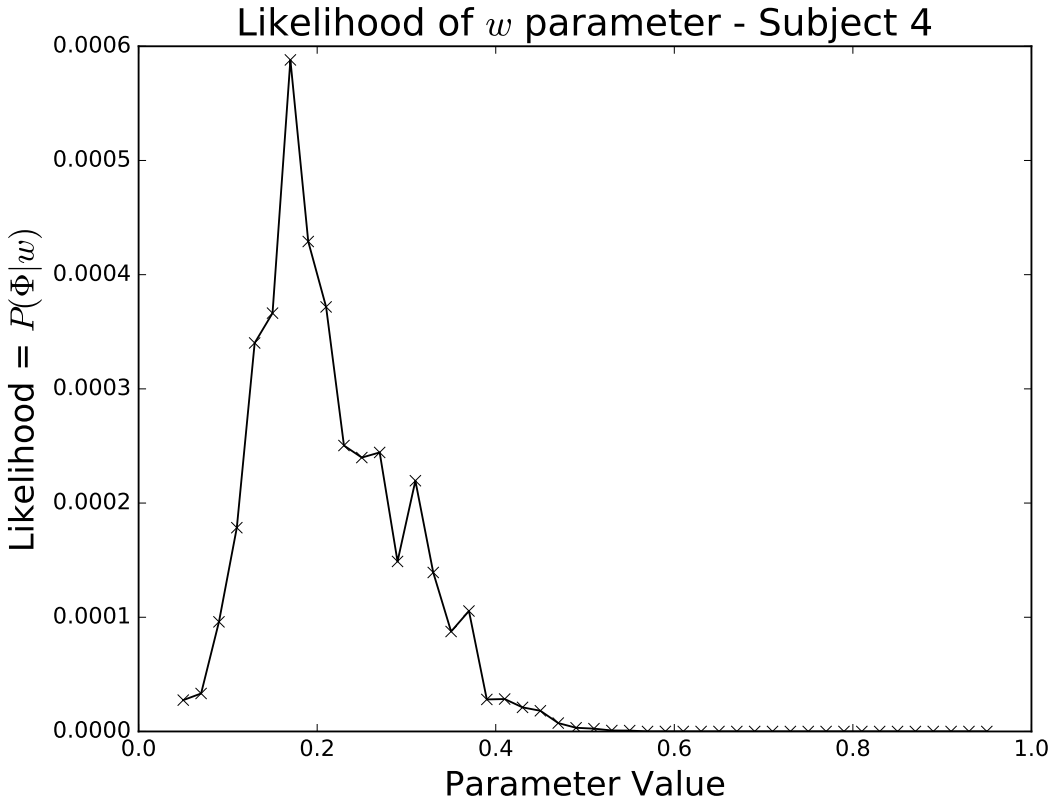

Figure 5.5: Plot of the computed likelihood of the w parameter for Subject 4. Observing the maximum value of the function, the parameter value can be determined to lie at roughly 0.2.

experience, but is unable to directly transfer through explicit programming. This allows the robot to learn the dynamics of a particular task faster, as it does not simultaneously need to learn the parameters controlling the overall strategy. This chapter specifically investigates a bimanual cleaning task where the task cost function controls the balance of the overall cleaning time and how much deformation the system is willing to impart on the part. Through experience, humans instinctively find a satisfactory balance when learning the task, but cannot convey the resulting strategy in a quantitative form to the robot. The core contribution of this work is an estimation algorithm that does not assume the demonstrator has full knowledge of the task dynamics (i.e., the material properties of the part being cleaned). This estimation with uncertainty applied to the demonstrator also allows for some level of filtering when multiple demonstrators are providing data. By looking for consistency between subjects, outliers can be detected who were using a different strategy entirely, without manual oversight or judgment that this is happening. This allows a robot to cleanly learn the overall task strategy, even as the human is learning the dynamics for a new task variation him or herself, which improves the overall system learning speed.

This chapter directly complements the previous imitation learning approach, as they focus on the transfer of different aspects of the task. Here, the overall strategy is estimated from the human, whereas the previous approach focused on generalization of trajectories. It is quite possible that certain tasks may contain both aspects in demonstrations that the robot can make use of. This possibility is discussed further as possible future research at the end of this dissertation.

Chapter 6: Online Learning of Part Deformation Models for Robotic Cleaning

This chapter is derived from work presented at the ASME Manufacturing Science and Engineering Conference (MSEC), 2016 [168].

6.1 Introduction

One task with high potential for robotic adoption is cleaning of objects and parts used in manufacturing and maintenance. Automatic robotic cleaning can cause substantial economic benefits in reducing labor costs associated with part finishing. Many manufacturing processes require post-processing operations to clean parts before they can be sent to the subsequent step in the process chain. The cleaning step is often necessary to remove the residue of chemicals used in the process (e.g., cutting fluids, lubricants, wetting agent) or debris (e.g., burrs). Many remanufacturing operations also require removal of the residue and buildup from the previously fabricated parts before reusing them in refurbished products. Tools used in manufacturing also require regular cleaning to keep them functional. Currently, robots are poorly suited to perform such tasks and thus significant human effort is required.

Depending upon the nature of the cleaning task, different modes of cleaning are used. In some situations cleaning fluid is applied to the surface to either dissolve the foreign particles or force the particles to separate from the surface. Many cleaning tasks require application of an oscillatory moving cleaning tool (often with abrasive particles embedded in the cleaning surface, e.g., sand paper) over the surface being cleaned. In this cleaning mode, undesired foreign particles are removed using mechanical scrubbing. This chapter is focused on cleaning tasks that are performed by the mechanical scrubbing actions. The scrubbing cleaning action is very similar to the one used in polishing and finishing operations. Hence the models and methods developed will be applicable to polishing and finishing tasks as well.

Automated cleaning of objects is challenging for several reasons. As the task inherently involves removal of unpredictable amounts of stain material, it is not possible to specify a planner that can be guaranteed to generate a trajectory that will solve the task in a single attempt. Continuous sensor feedback is needed to reevaluate the object status and recompute the cleaning actions given the most recent information. This factor alone eliminates the majority of existing robotic systems from being used as they must be programmed with particular motion paths with known effects. A further complication inherent to the task is that the robot must be able to access all surfaces on the part, which may not be possible from a single static pose of the part. Humans easily overcome this obstacle by coordinating the movements of two arms when cleaning and so a natural approach is to use a bimanual robotic setup, as seen in Fig. 6.1. However, this introduces additional complexity in coordinating the motion of the two arms.

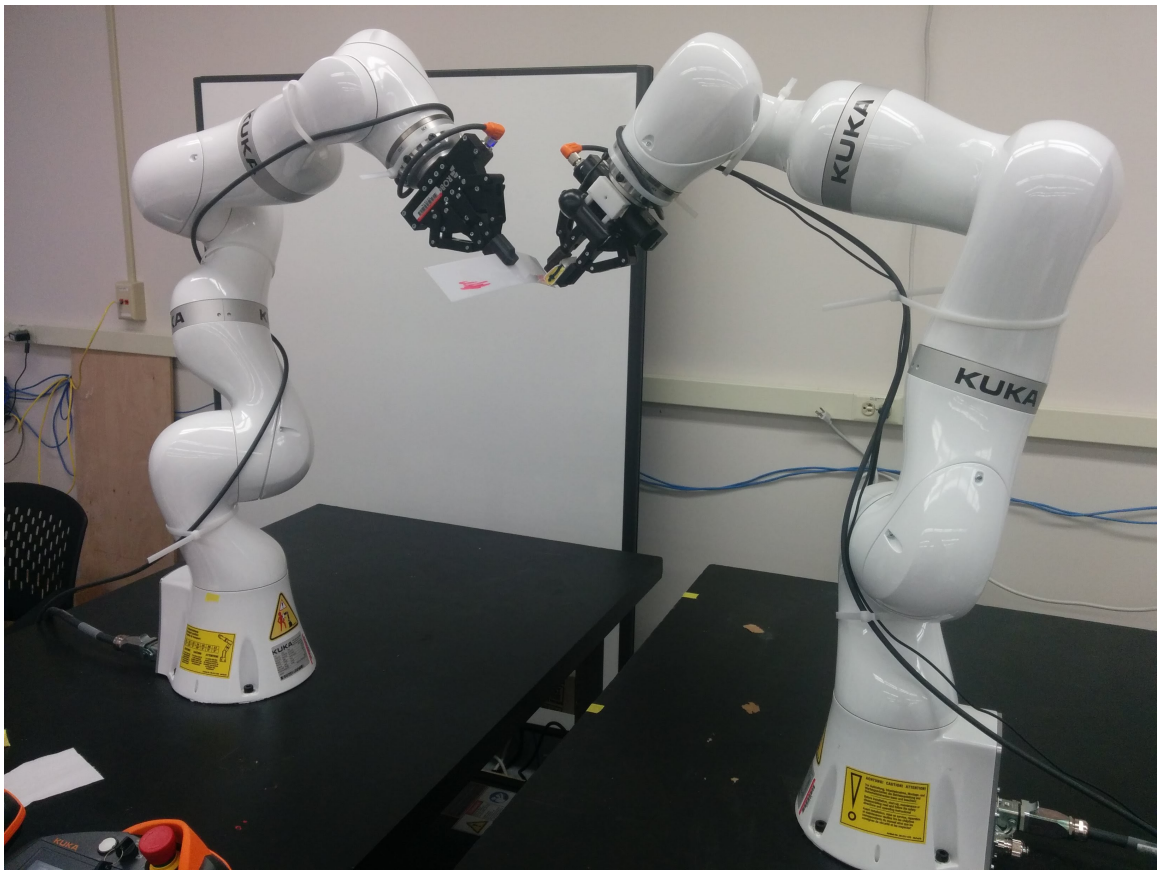

Figure 6.1: A workcell for bimanual robotic cleaning. The robot on the left is responsible for holding the parts and the one on the right uses a cleaning tool.

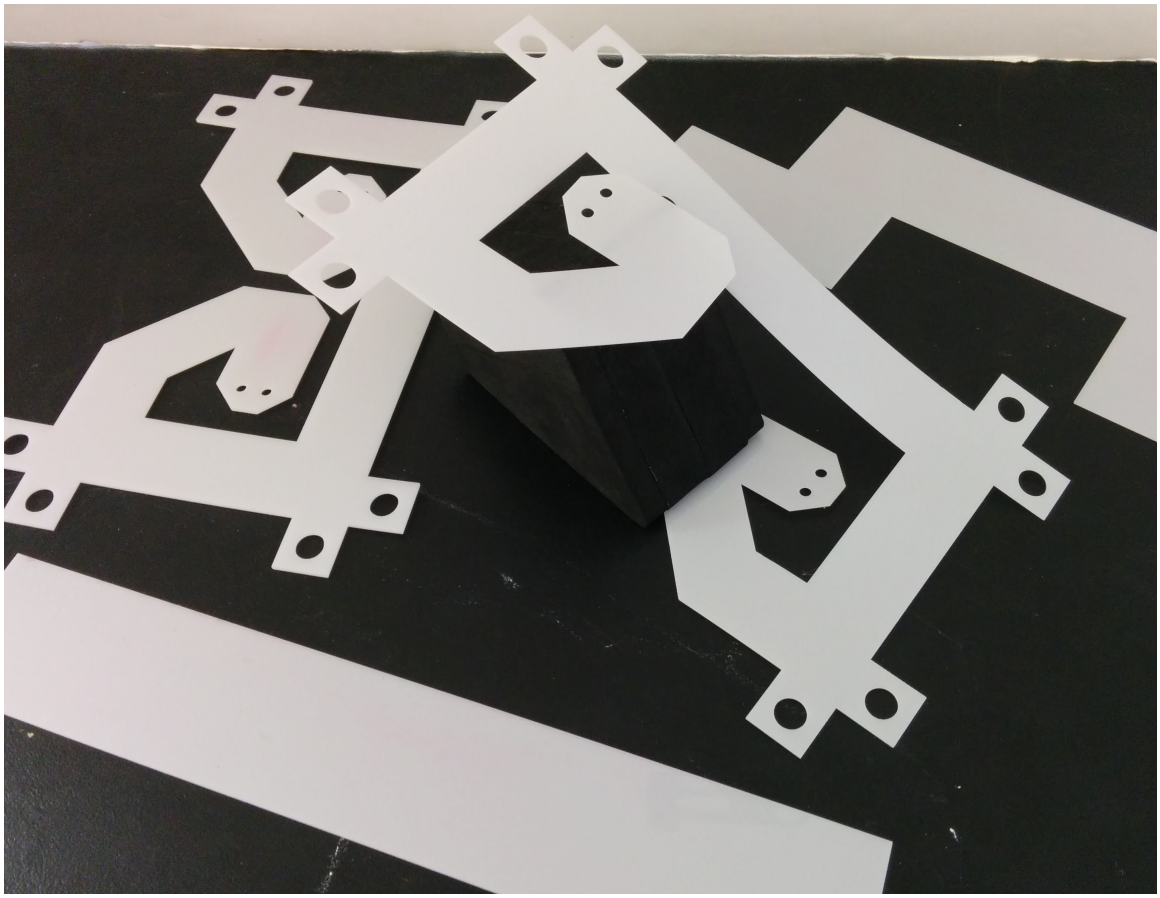

Figure 6.2: The three compliant parts used. Part A is the flat strip in the foreground, Part B is in the back-right, and Part C is centered and placed in a deformed position. The parts are made from 1/32" and 1/16" acetal.

Finally, the problem is further compounded when the objects to be cleaned are not rigid, meaning the interaction of the cleaning tool with the part alters the part geometry. Without precise knowledge of the part characteristics, including material properties, a planner cannot produce a trajectory that will be followed by the robot with minimal uncertainty. If the part is delicate, this is not acceptable, as deviation from a planned trajectory can easily damage the part permanently. Instead, the system will need to continually monitor the effect of its actions on the part and learn how to operate on it with maximum efficiency.

For example, consider the part shown in Fig. 6.3. Here, only a small force is being applied to the part but the part deformation is still large. In this case, it becomes less obvious how to plan for the two arms in order to minimize a cost metric such as overall cleaning time. Besides the Cartesian paths of the robot end-effectors, the force applied by the cleaning tool also becomes a key parameter controlling its performance. For such compliant objects, the force must be carefully monitored so as not to induce excessive deformations. This requires the robot to have some knowledge of how a part will react when an attempt to clean it with a particular set of planning parameters is made. The knowledge should be encapsulated in a model unique to each part. For simple parts, such a model can be efficiently derived from first principles or by fitting a small set of parameters to a test data set. However, for a fully generic approach, the model should be capable of general nonlinear behavior and should be learned and improved as the robot directly interacts with the part. To provide this ability, a Gaussian Process Regression (GPR) model is used for the part deformation model. As GPR is nonparametric, it is capable of approximating arbitrarily complex nonlinear function behavior, given sufficient data. Therefore, as the robot gains experience with different planning parameters, it concurrently improves its knowledge of how the part it is working on behaves.

This chapter presents an approach that is geared towards efficient cleaning of compliant parts. The primary contribution is to show that general nonlinear deformation models of various parts can be learned online and that the models can be exploited to rapidly optimize the plan trajectories of the two robot arms. This

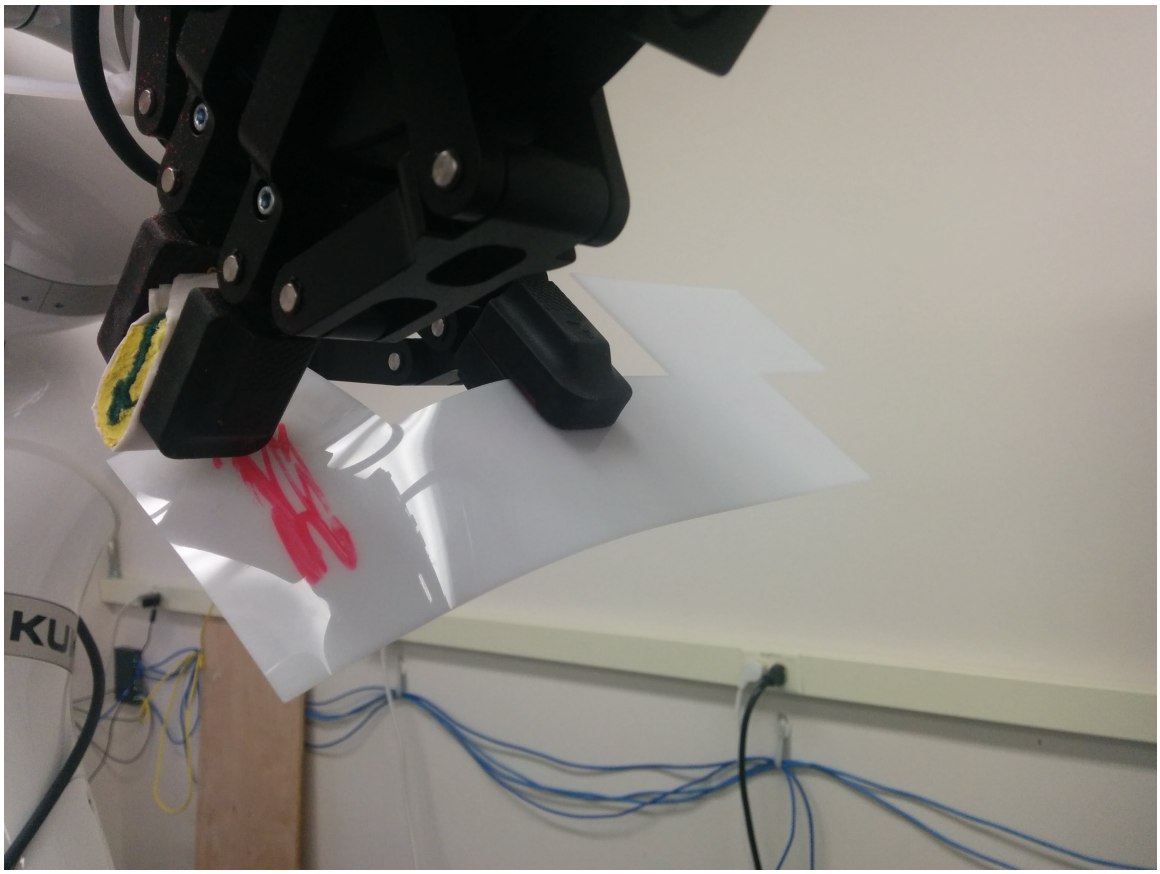

Figure 6.3: An example compliant part undergoing substantial deformation while being cleaned. The robot is commanded to apply 1N of force to the part surface.

includes how many different grasps are needed, where they are located, and how much force to apply to terminate the cleaning task in minimum time.

6.2 Approach

This chapter considers the cleaning problem in which the robot is given a part of known geometry with one or more convex regions on the surface where material must be removed. We say that the robot has completed a cleaning *episode* when the part has been fully cleaned. Each episode will be composed of one or more *attempts* where each attempt consists of the robot generating and executing a cleaning path with the goal of immediately terminating the episode. Each attempt will either finish successfully or will finish prematurely if it leads to unexpected excessive deformation of the part. In both cases, the part will be inspected for remaining material and if not deemed fully cleaned, another attempt will be performed.

In the system setup, the robot consists of two separate arms with one dedicated to holding and using a cleaning tool, and the other dedicated to holding the part up for the cleaning arm at various positions along its perimeter. This setup implies a significant time cost whenever the grasping robot has to change its grasp, as it must place the part down and pick it up again before the cleaning can continue. Therefore, it is highly desireable to generate plans for both the grasping and cleaning arms that minimize the number of grasps needed.

In the case of a rigid object, the plan can be determined simply from the kinematic constraints of the robot workspaces. For compliant objects, however,

the force applied by the cleaning arm will induce deformations, and the location of the grasp point is intimately related to their magnitude. For parts that have well-known geometry and material characteristics, methods exist to analytically determine deformations given the magnitude and point of application of forces, but in the general case it can be very difficult to predict in advance what deformation will occur. Such a scenario may occur with a robot in a rapid-prototyping manufacturing environment, where a robot may be regularly tasked to clean or polish newly created objects with complex geometries and/or material properties. In such settings, the robot should have the ability to learn a generic model of the object's deformation characteristics as it gains experience over time.

Therefore, the general approach relies on these models which are learned during the task episodes. Each model is specific to an individual task instance (i.e., object to be cleaned), which may be relevant for one or more task episodes, but not the entireity of the robot's experience. The models are then updated continuously as needed whenever new data is available.

6.2.1 General Problem Framework

Formulated generally, the robot runs a planner P for each attempt, which outputs a trajectory T for both arms, given the task instance model f_d and a task feature vector $\mathbf{x}$, containing the object geometry and the stained regions in the

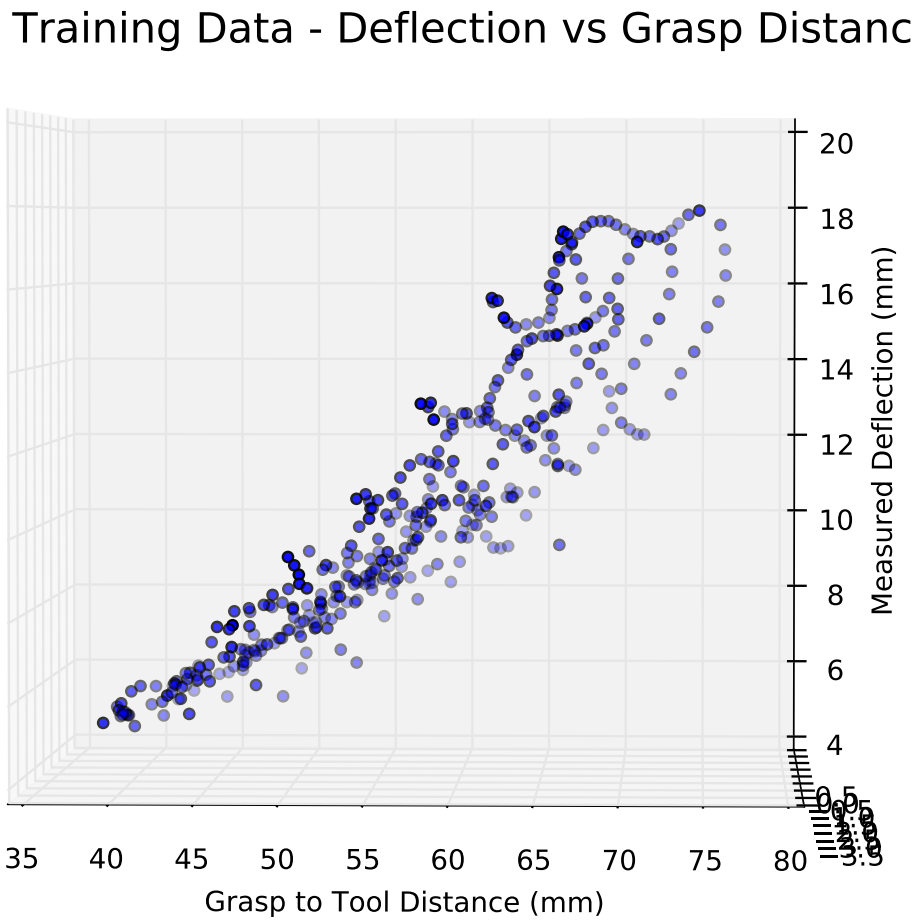

Figure 6.4: Example data obtained from a single task attempt. This data was taken from the second attempt in the first episode of the results seen in Fig. 6.9 and has been transformed into the features that this particular model is trained on: the distance between the grasp point and the cleaning tool, and the normal force applied by the tool. Here, the three-dimensional samples are shown projected onto the grasp distance and measured deflection axes. From this perspective, the average trend of the deflection vs grasp distance is clear.

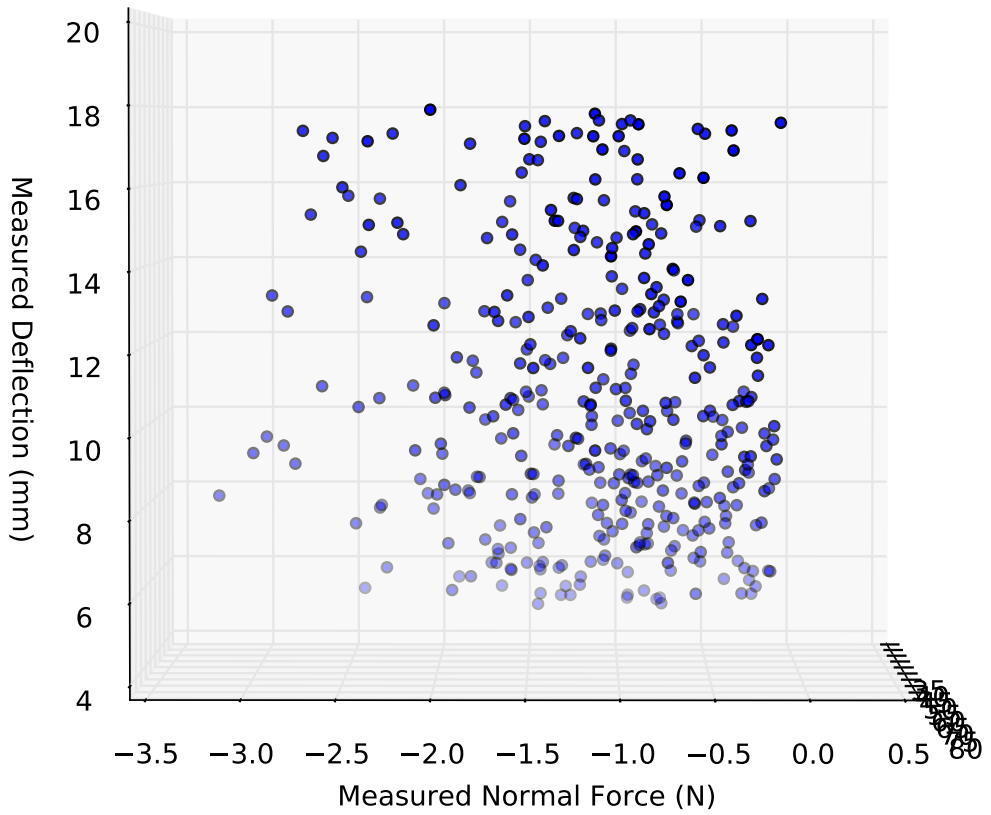

Figure 6.5: The same data as in Fig. 6.4 but projected onto the force and deflection axes. Here there is clearly much more noise than the case of the grasp distance as the force measurment is much more sensitive than position measurement. Gaussian processes are especially robust in the case where significant variation occurs in the data as they explicitly estimate the variance of the predicted output in addition to the expected value.

case of the cleaning task. The planner may be further parametrized by a manually defined parameter vector θ.

$$T = P_\theta(f_d, \mathbf{x}) \tag{6.1}$$

The trajectory is executed by the robot, constituting a single attempt. If further attempts are required for task success, the task feature vector is reestimated and the trajectory is recomputed. Each trajectory can be assigned some cost $J(T)$ such as the execution time. Therefore, the goal of the planner is to minimize both the per-attempt cost and the number of attempts per episode:

$$\min_\theta J_E = \sum_{i=1}^{N_A} J(T_i) \tag{6.2}$$

6.2.2 Learning Task Instance Models

The training data set consists of the tool and grasping positions, cleaning force, and part deflection that are recorded during the robot attempts. We fit a Gaussian Process Regression (GPR) model to this data. Gaussian processes are a smooth, nonparametric machine learning technique that has been widely used in robotics. We selected GPR for the model since it is capable of learning arbitrarily complex nonlinear functions given a sufficient number of data points and it is straighforward to implement. Further details on the behavior of GPR and its implementation can be found in Appendix A and the text by Rasmussen and Williams [76].

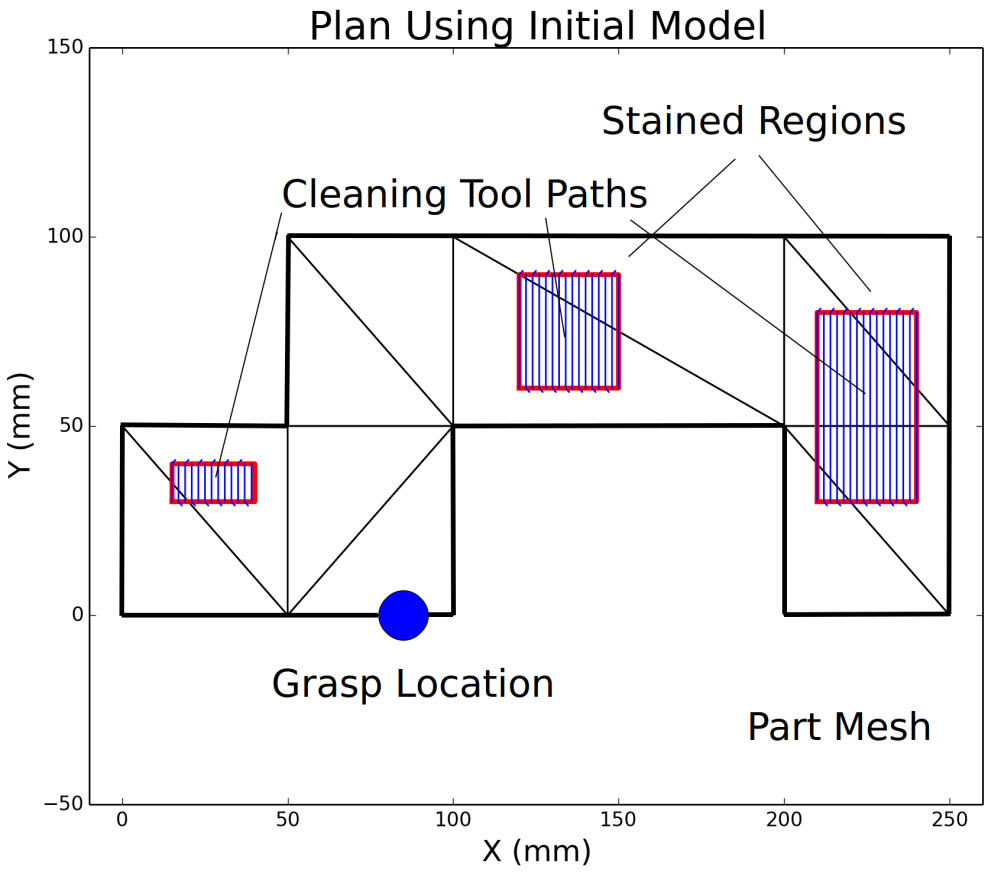

Figure 6.6: Initial plan generated to clean the part. The model predicts no deformation for all points in the stained regions and so only a single grasp is considered sufficient.

A Gaussian process in $\mathbb{R}^n$ is defined by a mean function $m : \mathbb{R}^n \to \mathbb{R}$, and a covariance function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, generally the squared exponential $K(x_1, x_2) = exp(-\frac{||x_1 - x_2||^2}{2l^2})$ (see A for additional details). The length-scale parameter l controls how strongly the data samples influence the model prediction of their neighboring points and is tuned using cross-validation or expectation maximization to best fit a batch of data. This then fully defines the task instance model $f_d(\mathbf{z}) \sim \mathcal{N}(m(\mathbf{z}), K)$, where $\mathbf{z} \in Z = \mathbb{R}^n$ is an element in a feature space chosen to represent the salient aspects of the task behavior. For the cleaning task, one general choice of Z could include both the location of the current grasp and the location of the cleaning tool, and a simpler alternative could be to have only the distance between the two.

When additional data points are available to train the model, there are two steps taken. First, a standard least-squares linear regression is run on the data and the resulting coefficients are used to define a linear mean function $m(\mathbf{z}) = c^T\mathbf{z}$. This is done to improve the model's ability to extrapolate to regions of the feature space Z where no data is present. If a zero mean function is used, then the model cannot capture a global trend in the data and the algorithm will have to explore the entire feature space before the model is fully accurate. Note however, that a linear mean function does not constrain the model in regions where data is available. The GP will smoothly move between approximating the acquired data samples and returning to the mean function in non-sampled regions of the feature space. Second, the model is retrained on the new data in addition to the previous data. Generally, the new data will be acquired in a unexplored region of the feature space, which improves both the local accuracy and the estimation of the overall trend.

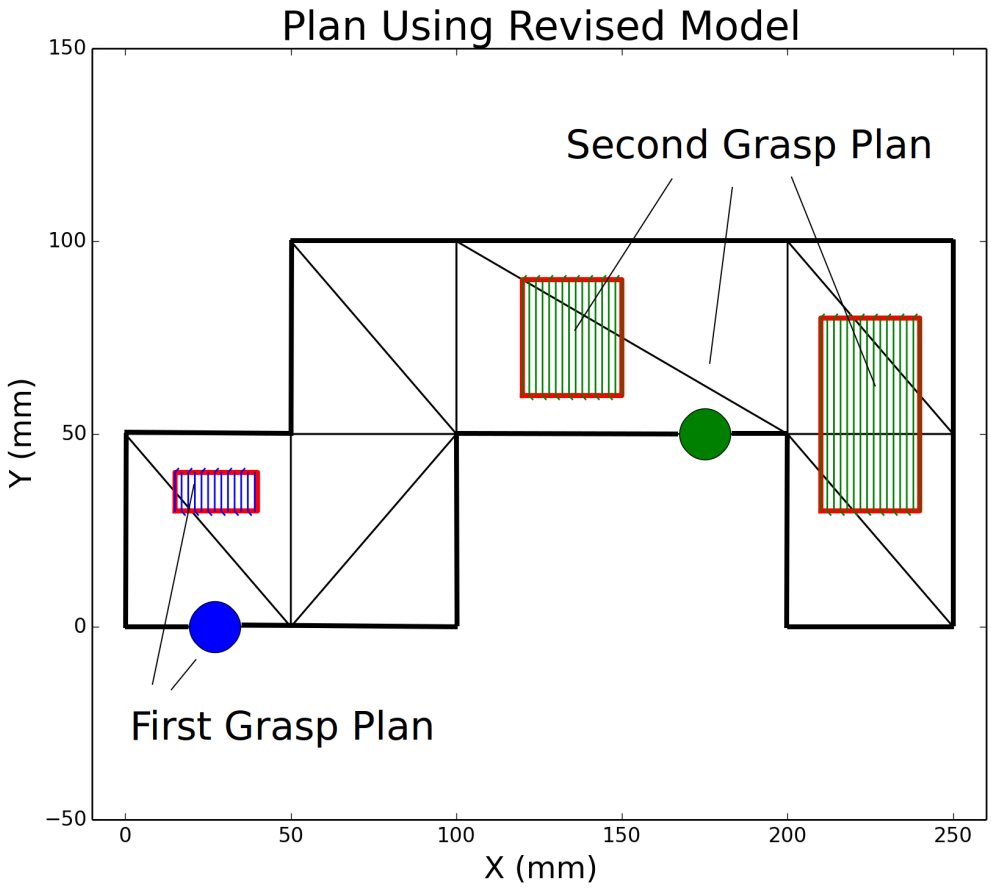

Figure 6.7: Revised plan using the known part deformation model. Two grasping locations are required to keep the maximium expected deflection below the threshold. The segments covered by the cleaning tool are shown in the same color as their corresponding grasp location.

See Figs. 6.4-6.5 for an illustrative example set of data used to train the model. In this case, the model is in two dimensions and the features are the aforementioned distance between the grasping point and the cleaning tool, and the force applied by the tool.

6.2.3 Coverage Planner

To generate the tool paths over the stained regions, a simple coverage planner is used that moves the cleaning tool in a set of parallel linear passes to cover the regions identified by the perception algorithm. The tool passes were all set parallel to the y-axis of the part, and the distinct regions provided by the perception algorithm were all planned separately. This method is certainly not an optimal method of covering the stain regions in all cases, but is useful for illustrating the general learning approach. In the general case, the approach will also work effectively with any standard coverage planning algorithm.

The planner was parameterized with two values: the normal force applied by the tool to the part surface and the spacing of the tool passes over the regions detected to be stained. These parameters nicely provided complementary influences on the system behavior as the tool force primarily affects the deformations of the part, and the tool pass spacing is a significant factor in the overall cleaning time.

Given a particular set of values for the planner parameters, a particular trajectory can be generated composed of points on the part surface: $T = \{(x_1, y_1)...(x_n, y_n)\}$. The system then optimizes the grasp locations for the trajectory (see Alg. 6). This is done iteratively over the number of grasp positions. Each grasping position is defined by a single real-value describing the distance along the part's perimeter, excluding lengths where a grasp is unachieveable due to robot or gripper constraints. Multiple grasp positions are therefore represented by a point in $\mathbb{R}^N$, with N different grasps. Starting with $N = 1$, a objective function is defined where each point

$p_i = (x_i, y_i) \in T$ is associated with the closest grasp point $g_c \in \mathbb{R}$. The function then computes the average expected deflection of each point using the current part model deformation model. In general, the model can be defined to make use of an arbitrary number of inputs, such as the absolute positions of both the point on the part and the grasping location, geometrical features unique to the part, or other planner parameters. For simplicity the initial implementation used just two inputs: the cleaning tool force and Euclidean distance between the cleaning point and the grasping point. The optimal set of grasps $\mathbf{g} \in \mathbb{R}^N$ is found using a standard non-linear optimizer that minimizes this objective function over the range of possible grasp points. The optimal value is then tested for its *maximum* expected deformation. If it exceeds a given threshold, the process is repeated after incrementing N. Otherwise, the planner terminates sucessfully.

An illustration of two planner outputs can be seen in Figs. 6.6 and 6.7 where an example set of stains is displayed on a triangular mesh of one of the test parts. In Fig. 6.6, the part deformation model is newly initialized and so all predicted deflection under any force are zero. Therefore, the planner finds only a single grasp point and all the tool passes are mapped to the single grasp. In constrast, Fig. 6.7 shows the same part and stains, but with a deformation model with prior knowledge. Here, the planner finds no single grasp where the predicted deflection is below the given threshold and so it is forced to search the space of two grasps.

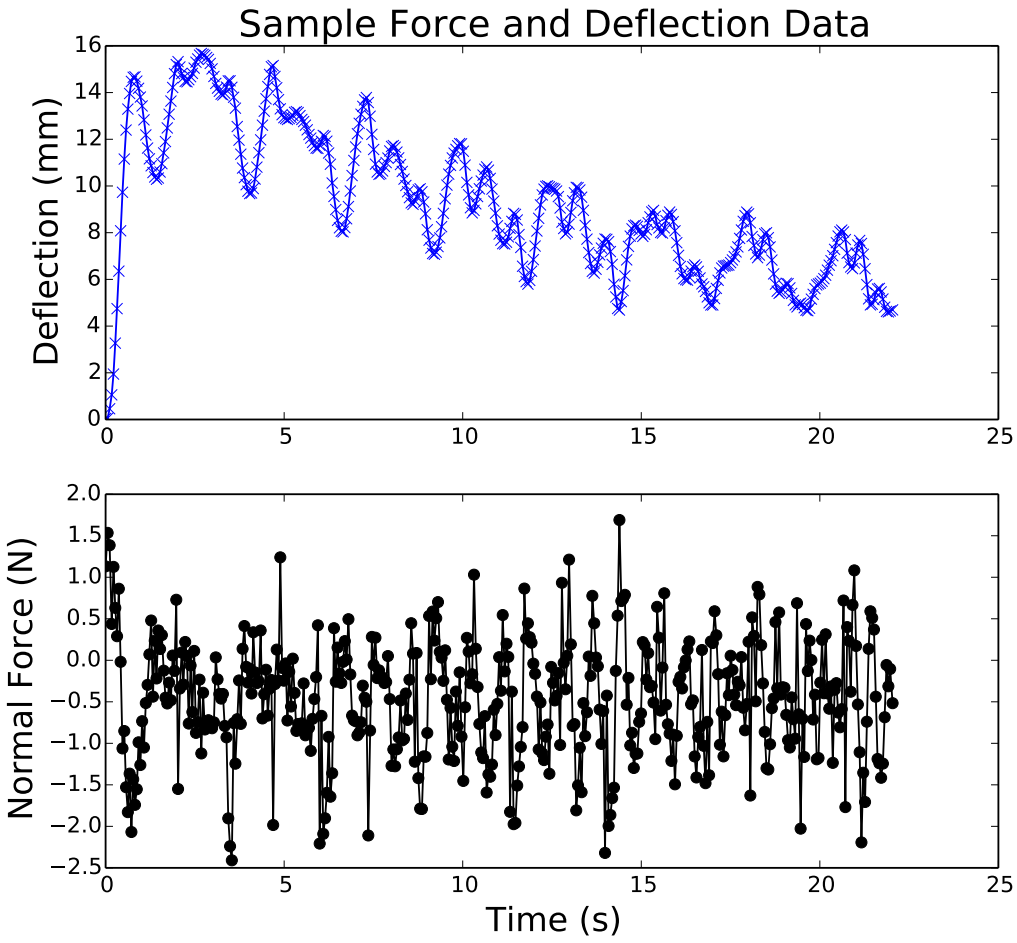

Figure 6.8: Example force and deflection data as reported by the robot. This particular data was collected during execution of the third iteration in Episode 1 of Part B, seen in Fig. 6.11

6.2.4 Planner Parameters

As the planner parameters θ can significantly affect the overall cost of a task episode, it is useful to also learn a policy where the optimal parameters can be found. There has been substantial work in the reinforcement learning community to develop such policy learning algorithms [169]. For simplicity, this work only focuses on learning the task instance models and uses simple heuristics to obtain the planner parameters. However, other implementations may use more sophisticated techniques without issue in this framework as the model for the planner parameters is independent of the task instance models.

In the approach for the cleaning task the two parameters used were the normal force applied by the cleaning tool and the spacing between cleaning passes. The tool spacing was calculated separately from the rest of the system when given a particular set of stains so that the cleaning time would not exceed a given threshold. An initial force value was set to 0.5 N, experimentally determined to cause minimal deflection in the part. Then for an attempt, if the maximum deflection threshold was not exceeded and stained regions remained, the force level was incremented by 0.5 N. However, if the threshold was passed and the attempt terminated early, the force level was maintained and new grasp positions were computed from the improved deformation model. The force amount was never decreased as long as stained regions remained. A maximum number of allowed grasps was set to prevent the system from infinitely attempting to use more grasps when the true deformation characteristics

of the part make it impossible to apply the desired force without exceeding the threshold.

When implementing the system to apply the normal force, it is important to consider the dynamic behavior of the cleaning robot and provide some safeguards against dangerous behavior. Such behavior can arise, for example, if the system has very low certainty about the expected stiffness of the part, but directly applies the full cleaning force immediately at the beginning of a cleaning attempt. This could lead to an extreme deformation and possible permanent damage of the part if the actual stiffness is much lower. In our implementation, we have the ability to gradually apply the normal force and halt the cleaning attempt if the observed behavior is dramatically different than what was expected. More advanced systems can reduce the need for this gradual application by coupling it to the current model uncertainty, which would allow the robot to clean faster and be more bold with a well-known part, just as humans naturally do.

6.2.5 Model Improvement

As the robot executes the path computed by the planner, it pushes the cleaning tool surface against the part with the pre-selected force. However, the actual force value applied on the part can vary substantially, due to the transient dynamics of the deflection behavior. Using its built-in joint torque and position sensors the robot can report the actual force and position seen by the cleaning tool at a fast update rate (up to 100 Hz). These values are recorded during execution of the cleaning

Algorithm 6 Cleaning Task Attempt Planner

Inputs: Task parameters ($\mathbf{x} \in \mathbb{R}^n$), Current part model ($f_d$), Task specification (stained regions) ($\sigma \subset \mathbb{R}^2$), Possible grasp parameters ($G \subset \mathbb{R}$)
Outputs: Cleaning tool trajectory ($T = \{(x_1, y_1), \ldots, (x_n, y_n)\}$), Grasp location parameters ($(g_1, g_2, \ldots, g_N)$)

function PLAN($\mathbf{x}$, f_d, σ, G)
  $\mathcal{P} \leftarrow \varnothing$
  $T \leftarrow P_{\mathbf{x}}(\sigma)$ (parameterized coverage planner)
  $N \leftarrow 1$
  while $N < N_{max}$ **do**
    $c_f(\mathbf{g}) \triangleq$ GRASPCOST($f_d, T, \mathbf{x}, \mathbf{g}$)
    $\mathbf{g}^* \leftarrow optimize(c_f(\mathbf{g}))\ for\ \mathbf{g} \in G^N$
    $d_{max} \leftarrow max(\{f_d(p_i, \mathbf{x}, \mathbf{g})\ \forall\ p_i \in T\})$
    if $d_{max} < d_{thresh}$ **then**
      $\mathcal{P} \leftarrow T, \mathbf{g}^*$
    else
      $N \leftarrow N + 1$
    end if
  end while
  return $\mathcal{P}$
end function

function GRASPCOST(f_d, T, $\mathbf{x}$, $\mathbf{g}$)
  $D_{pred} \leftarrow \{f_d(T_i, \mathbf{x}, \mathbf{g})\ \forall\ T_i \in T\}$
  $\hat{d}_{avg} \leftarrow mean(D_{pred})$
  return $\hat{d}_{avg}$
end function

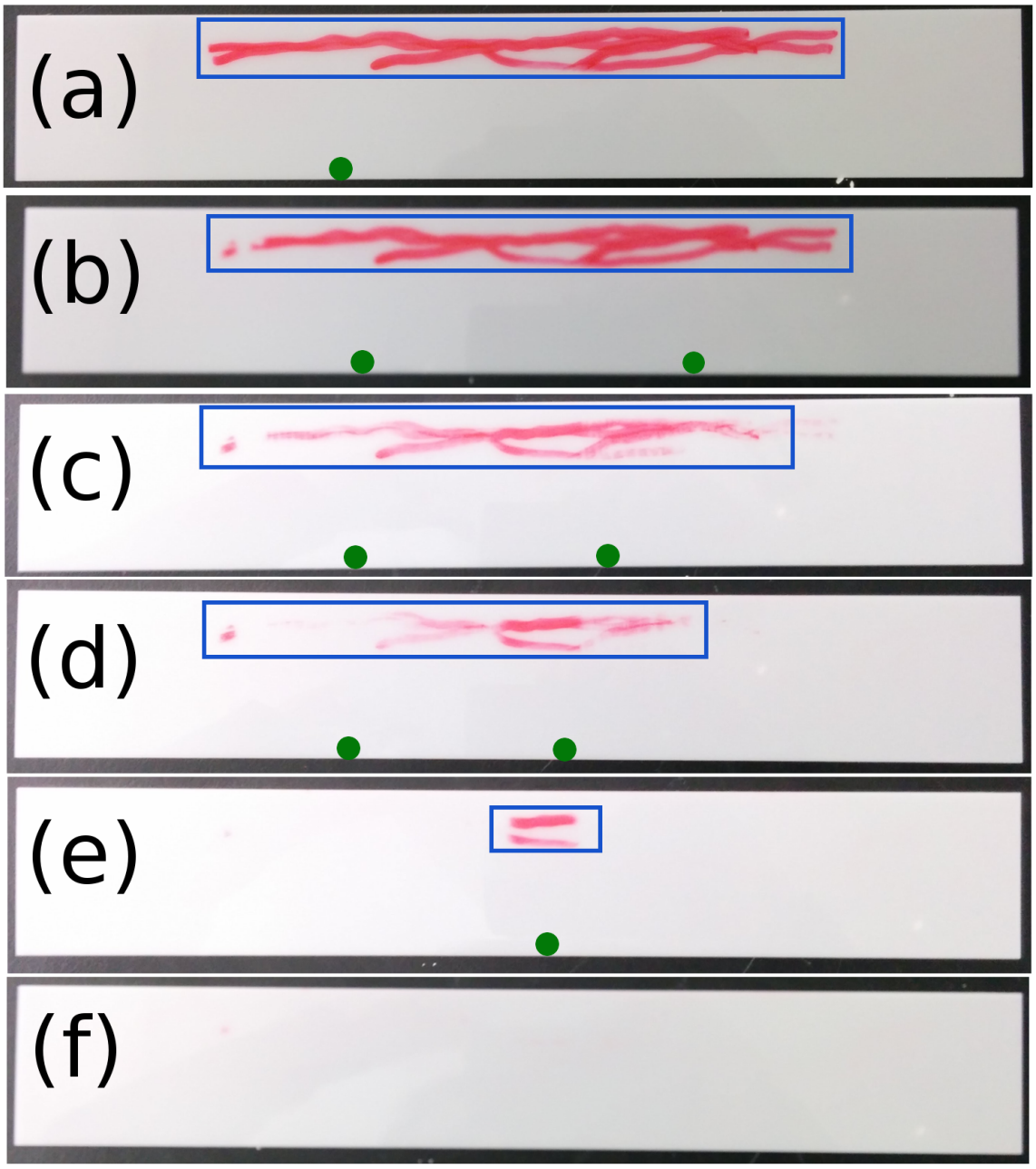

Figure 6.9: Part A - Episode 1. Example result of learning the deformation model of a simple rectangular part. The blue boxes are the regions given to the planner to cover and the green points are the grasp point(s) selected by the algorithm. The learning algorithm has no prior knowledge of the part behavior and therefore selects an arbitrary grasp point to start with in (a), expecting zero deflection everywhere. The execution immediately fails and the resulting data is sufficient for the planner to require the use of two grasp points in (b). Notice that the planner continues to use two grasp points as the cleaning region shrinks in (d), because the cleaning force is rising, until (e) when two grasp points are redundant.

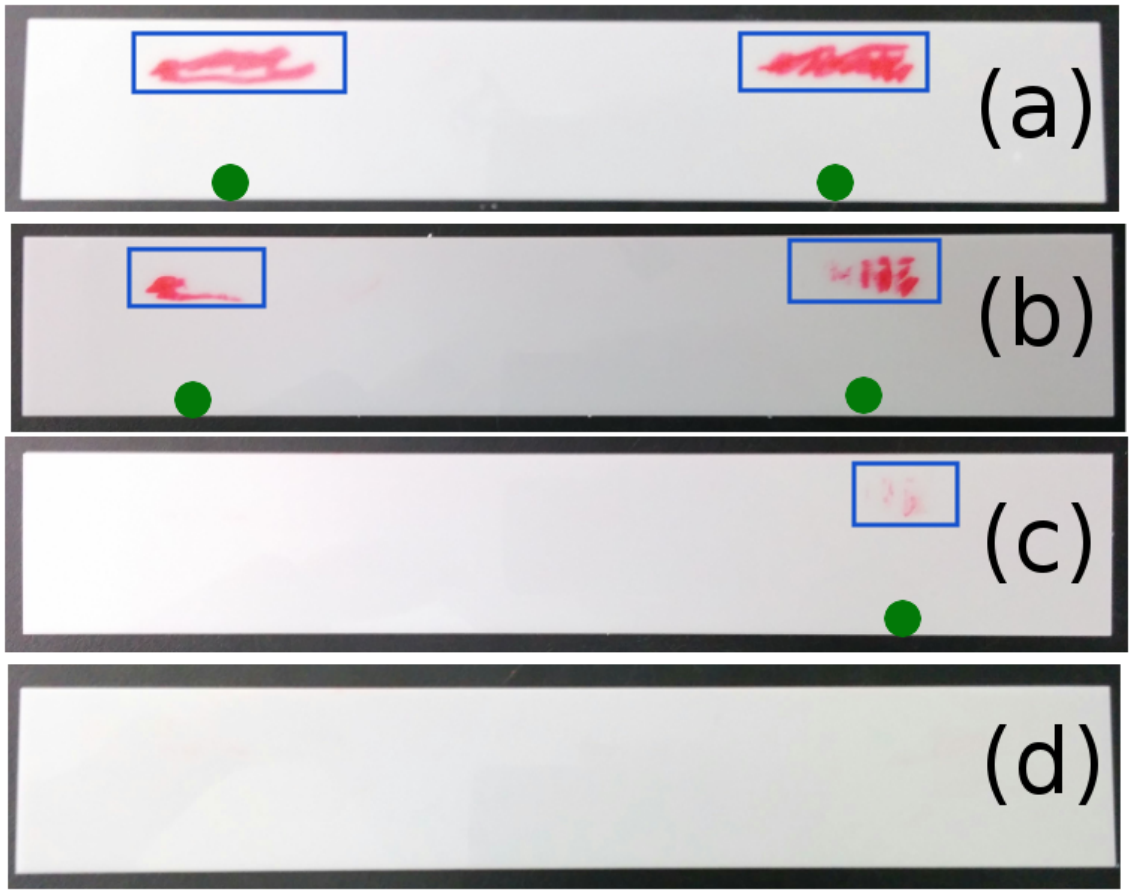

Figure 6.10: Part A - Episode 2. In this episode, the model learned from episode 1 is reused by the planner from the first iteration. Therefore, the algorithm immediately requires the use of two grasp points and the overall cleaning time is shortened.

path and the resulting samples are used to improve the deformation model. Figure 6.8 shows the raw data from a cleaning attempt of a small region, executed in 22 seconds. The data contains substantial noise, primarily resulting from the cleaning tool moving between regions of high and low stiffness, and the resulting absorption and release of strain energy as the part shape deforms. This data is filtered to include only the samples where a negative force is exerted on the cleaning tool, implying the part surface has been positively deflected and is exerting a reaction force on the tool. The filtered points are then added to the collection of previous samples and the model is then retrained on all data samples.

The retrained part model is then used in planning for the next task attempt. After the part model update after each attempt, the grasp planner is given more information to determine the number of grasps needed, which may be more or fewer than the previous attempt, depending on what amount of stained regions remain. This leads to a fast determination of the optimal number of grasps and their locations to obtain task success.

6.3 Results

6.3.1 Parts

Results are shown using the online learning approach for a set of planar parts made from thin plastic. The parts used are visible in Fig. 6.2, which also illustrates their deformation characteristics. Initial results were obtained on a set of three parts: a 300×50 mm rectangular strip (Part A) for illustration of the grasp planning, a

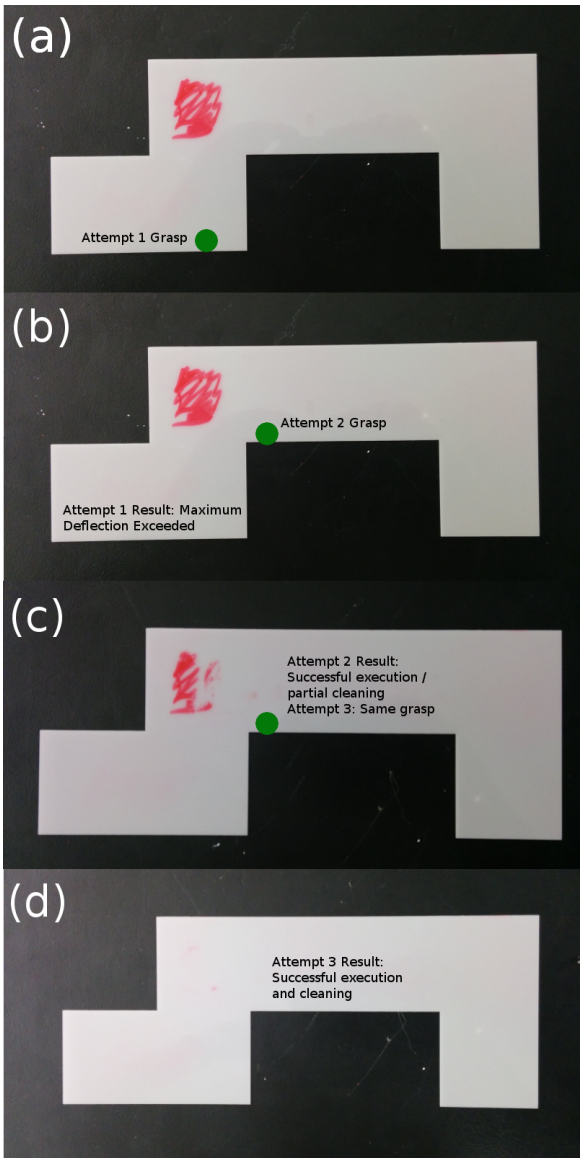

Figure 6.11: Part B - Episode 1. Example result of learning the deformation model of a rectangular part with cuts. Here the learning algorithm again is given no prior information about the deformation model and three attempts are needed to clean the part successfully. Attempt 1, shown in (a), terminated with negligible cleaning action due to the unknown prior model but the data collected was sufficient to update the model for attempt 2, which finds a closer grasp point, seen in (b). This is enough to enable successful execution of the cleaning plan, but the force is insufficient to completely clean the stain. The cleaning force was then increased and the part was cleaned successfully after attempt 3.

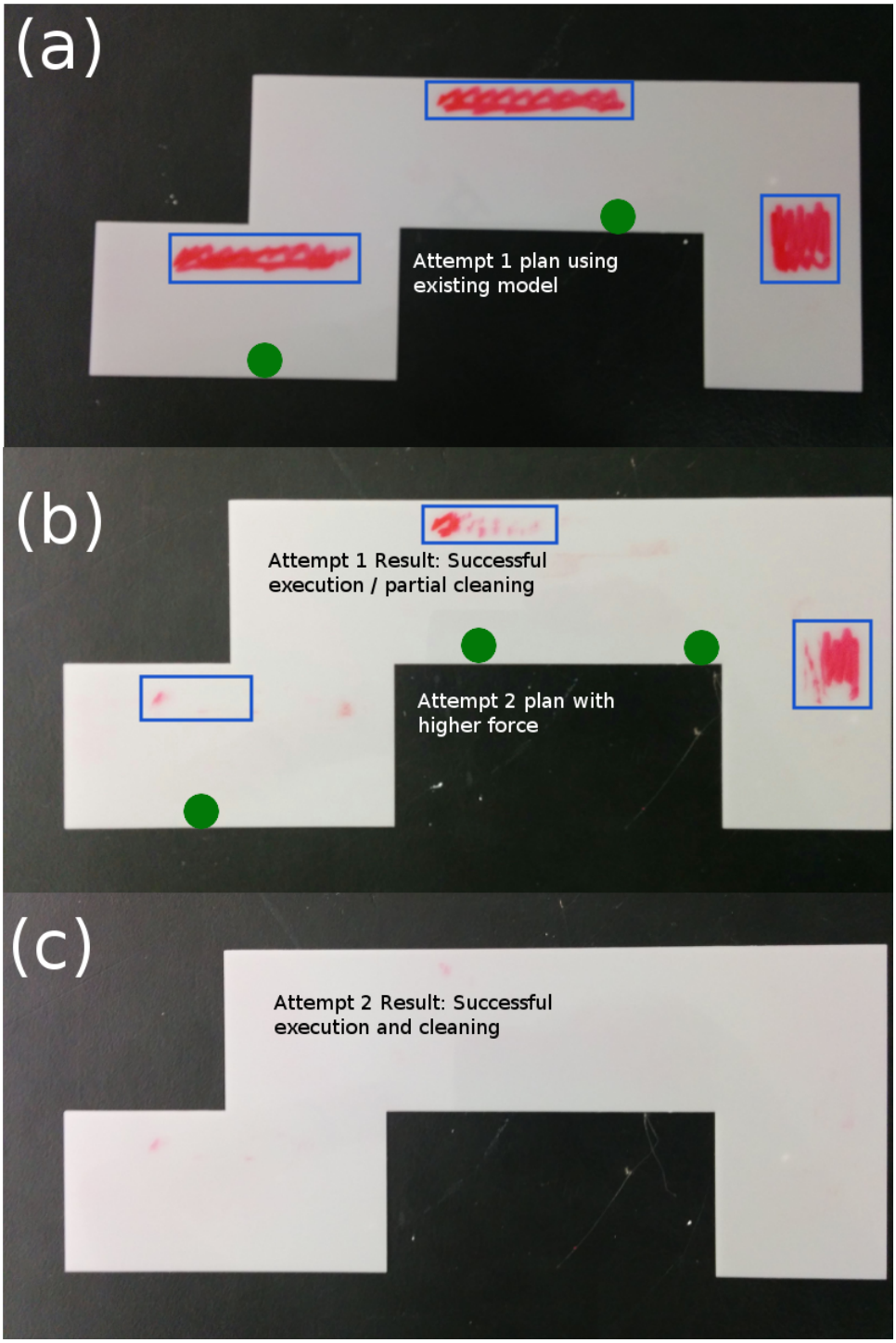

Figure 6.12: Part B - Episode 2. The episode uses the previously learned model from the first episode in the previous figure. This result illustrates that data collected from a single region to be cleaned and nearby grasp points is sufficient to generalize to other regions. The planner selects two grasps for the first attempt in (a), which executes without exceeding the deflection threshold. For the second attempt in (b), note that although the previous attempt executed successfully, the planner still calls for the use of three grasp points. This is because the cleaning force has been increased and the system is able to anticipate that the previous two grasps are insufficient. The part is cleaned successfully after the second attempt.

250×100 mm rectangular part with various cuts (Part B), and a more complex part designed to resemble an actual component used in a manufacturing setting (Part C). For simplicity of demonstrating the learning method, in parts A and B, only the bottom edges of the parts were given as feasible grasp positions.

For all parts, the features of the learned deformation model consist of the normal force applied by the tool and the distance between the tool point-of-contact and the grasping location. For parts A and B, this distance is calculated using the standard two-dimensional Euclidean distance metric. For Part C, given its winding geometry, the distance is computed along a the centerline running through the length of the part.

The stains were created with a highly visible marker and were calibrated to require a nontrivial amount of force to remove. The initial cleaning force was given as 0.5 N and was increased in increments of 0.5 N. The maximum deflection specified for all parts was 30 mm.

6.3.2 Stain Detection

A color based detection system is used to determine which regions of the part were dirty before each iteration. An overhead camera captured an image of the part placed flat on a support. K-means clustering was run on the pixels of the image to classify the image into clean and dirty regions. Adjoining pixels classified as dirty were then surrounded by bounding rectangles. Any rectangles closer than a specified

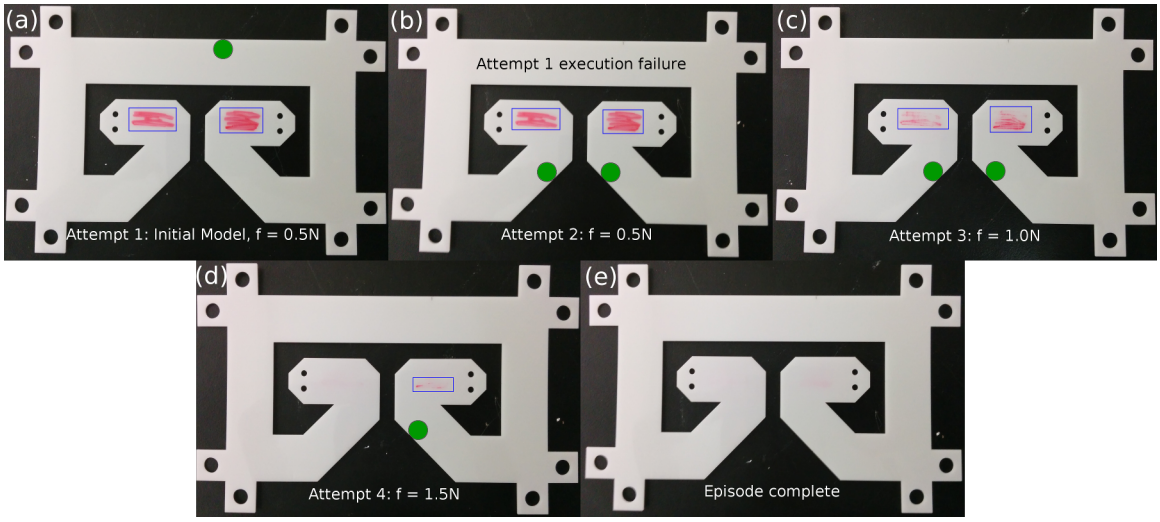

Figure 6.13: Part C - Episode 1. Here the algorithm is enable to learn a model quickly even on a fairly complex part. With no prior data, the planner selects a poor grasp point in (a), which leads to a failed attempt from excessive deflection. Using the distance metric of measuring along the centerline of the part, the resulting model improvement is sufficient to require two grasps for the following three attempts.

threshold were merged to provide the planner with a simpler input of a few larger regions rather than many small ones.

6.3.3 Physical Experiments

Preliminary results show successful fast convergence on the test parts with different stain configurations. Each part was subject to two cleaning episodes, with the deformation model learned in the first episode carried over to the second. To show the performance of the system, in the first episode an initial model of the part deformation is provided that assumes no deflection will occur regardless of force, which implies free choice of grasp for the algorithm. An example result for Part A is shown in Fig. 6.9, with the part status shown after each attempt. This demonstrates how the planner generates optimal grasp locations quickly after just

one cleaning attempt. The episodes for Part B are seen in Figs. 6.11 and 6.12 and for Part C in 6.13 and 6.14. These results show the previously learned deformation can be used to correctly predict grasps for new stain configurations. Overall, the results show fast cleaning performance even when initialized with no knowledge of the part deformation behavior, and more rapid performance when a previously learned model is available. Roughly 3-4 cleaning attempts are needed on average to build an accurate model approximation for a new part. Using the previously learned model allowed one fewer iteration on all parts.

6.4 Summary

This chapter presents an approach for bimanual robotic cleaning of compliant objects. The robot system has no initial information about the deformation properties of the object to be cleaned and it incrementally learns a black-box deformation prediction model through attempts to clean. Using the currently available model, a planner optimizes the grasping locations on the part to simultaneously minimize the deflection during cleaning and the overall cleaning time. Results show that the robot is able to learn a sufficiently accurate model fast enough to only require a few cleaning attempts. Furthermore, the learned model persists for the same part across multiple cleaning tasks, helping to further shorten the needed cleaning time. The primary contribution of this approach is a method that couples direct prediction of the task dynamics to a planner that searches for the overall best task execution strategy. By using both components together, the need for a fully accurate task

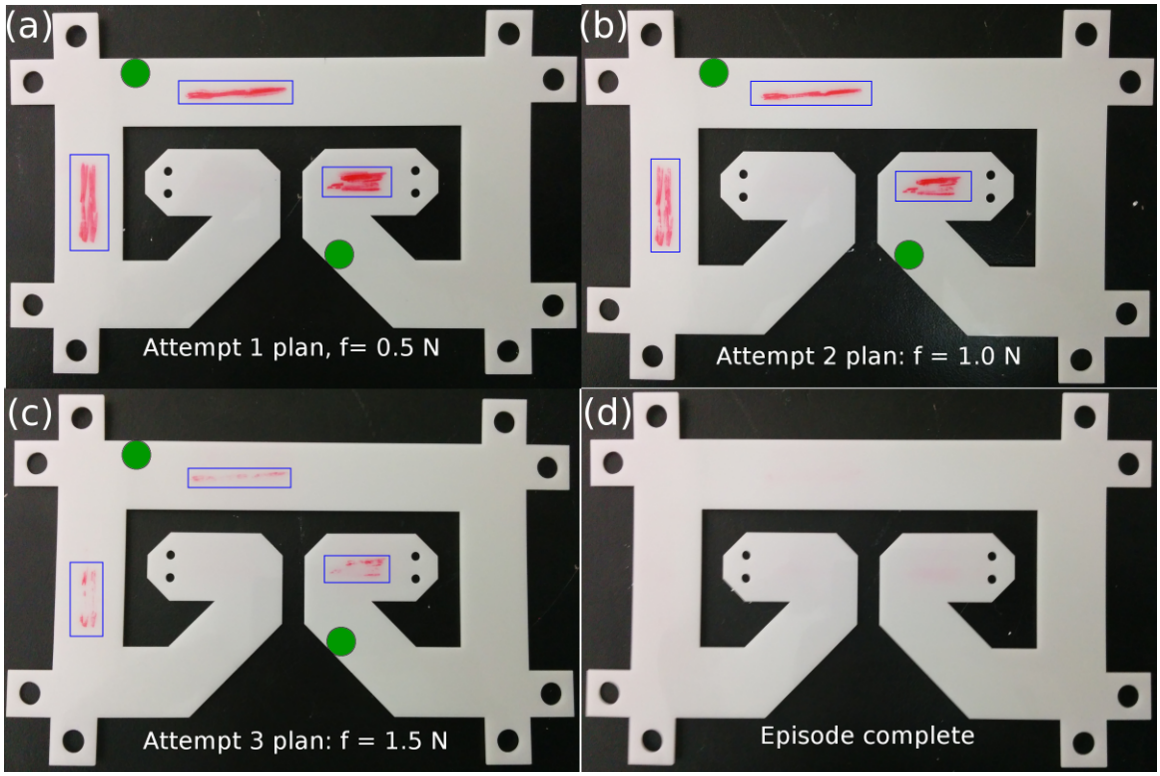

Figure 6.14: Part C - Episode 2. Here, the planner is given the learned model from the episode shown Fig. 6.13 and a new set of stains. The model successfully predicts that a single grasp point is needed for the stains on the top and left and it is able to finish cleaning the part in three attempts.

dynamics model is reduced and the updates to the model gained from just a few task attempts becomes sufficient to find the optimal execution strategy.

Chapter 7: Integration of Deformation Model Estimation with Planning for Robotic Cleaning of Elastically-Deformable Objects

This chapter is partially derived from work published at the International Conference for Automation Science and Engineering (CASE) 2016 [170]

7.1 Introduction

As robots become more prevalent and capable in manufacturing settings, they will be required to perform more complex and non-repetitive tasks that require online learning and adaptation, such as bin-picking [171] and assembly [172]. An example of such a task is cleaning and finishing of parts used in manufacturing. The majority of manufacturing settings require some form of cleaning or polishing of new parts and similar tasks are abundant when performing maintenance. Currently, robots are poorly suited to perform such tasks and thus significant human labor is required instead.

Automated cleaning of objects is challenging for several reasons. As the task inherently involves removal of unpredictable amounts of stain material, it is not

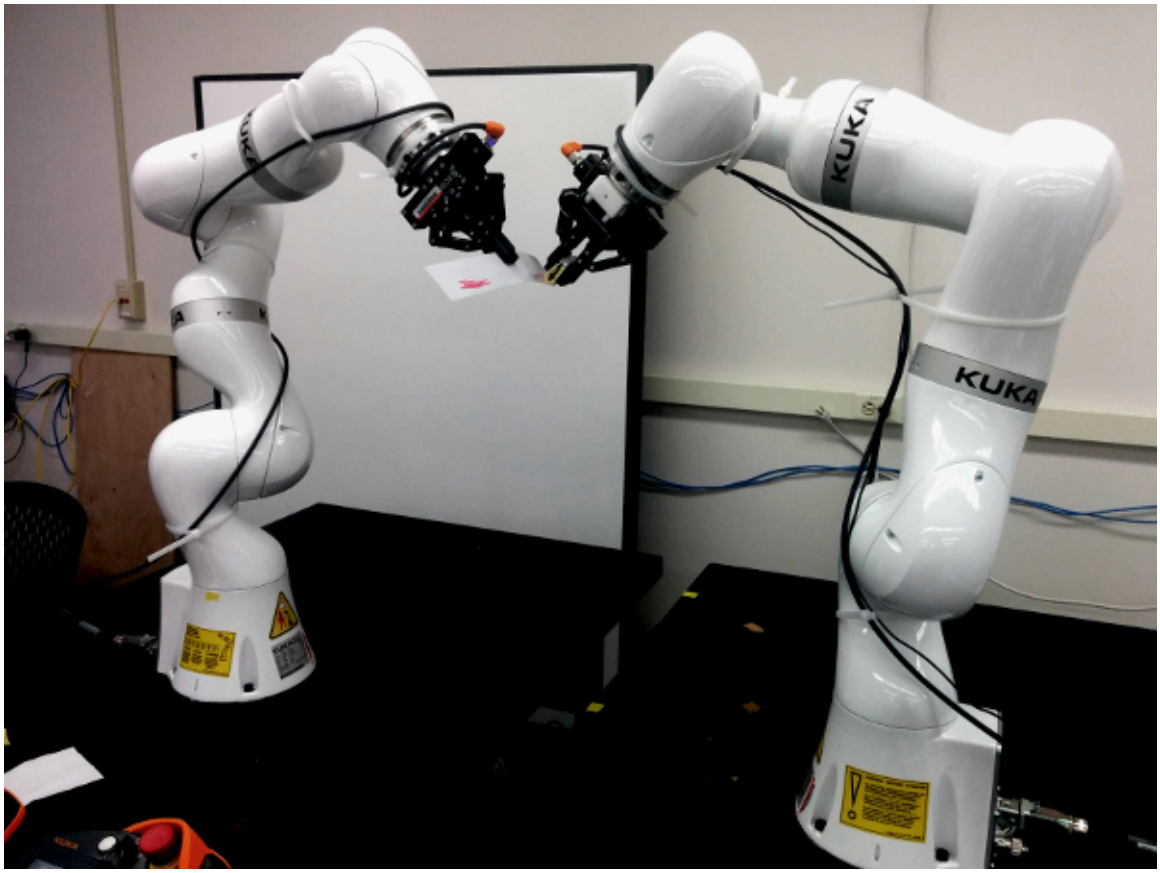

Figure 7.1: Bimanual robot cleaning setup. The cleaning (active) arm on the right holds and moves the cleaning tool, while the grasping (supportive) arm on the left holds the part.

possible to specify a planner that can be guaranteed to generate a trajectory that will solve the task in a single attempt. Continuous sensor feedback is needed to reevaluate the object status and recompute the cleaning actions given the most recent information. A further complication inherent to the task is that the robot must be able to access all surfaces on the part, which may not be possible from a single static pose of the part. Humans easily overcome this obstacle by coordinating the movements of two arms when cleaning and so a natural approach is to use a bimanual robotic setup, as seen in Fig. 7.1, where an *active* arm performs the cleaning while a *supportive* arm holds the part.

Finally, the problem is further compounded when the objects to be cleaned are not rigid, meaning the part geometry can change over time, depending on the particular actions performed by the manipulator arms. Without precise knowledge of the part characteristics, including material properties, a planner cannot produce a trajectory that will be followed by the robot with minimal uncertainty. This is not acceptable if the part is delicate, as deviation from a planned trajectory can easily damage the part permanently. Instead, the system will need to continually monitor the effect of its actions on the part and learn how to operate on it with maximum efficiency.

With parts of low stiffness, where even small forces can cause large displacements, it becomes less obvious how to plan for the two arms in order to minimize a cost metric such as overall cleaning time. Besides the Cartesian paths of the robot end-effectors, the force applied by the cleaning tool also becomes a key parameter controlling its performance. For such compliant objects, the force must be carefully

monitored so as not to induce excessive deformations. This requires the robot to have some knowledge of how a part's geometry will change when an attempt to clean it with a particular set of planning parameters is made. The knowledge should be encapsulated in a model unique to each part. For simple parts, such a model can be efficiently derived from first principles or by fitting a small set of parameters to a test data set. However, for a fully generic approach, the model should be learned and improved as the robot directly interacts with it. To provide this ability, we use a simplified finite element method model that can approximate the behavior of a part with continuously varying material properties, and learn the parameters of the model incrementally as the robot interacts with the part.

To complement the full learning system, a separate model of the cleaning tool behavior is learned and improved in parallel with the deformation model of each part the robot interacts with. This model describes the elements of the cleaning behavior that are agnostic to the part being cleaned. In particular, for a given cleaning tool, part material, and stain type, a tool performance model can be learned that will apply to all parts of that material. Having an accurate tool performance model is valuable because it enables the selection of parameters that will minimize the cleaning process cost.

To summarize, this chapter presents an approach that builds towards a general framework for efficient cleaning of compliant parts. The primary contribution is to show that general deformation models can be learned online with minimal prior knowledge and that the models can be exploited to rapidly optimize the plan trajectories of the two robot arms. This includes how many different grasps are needed,

where they are located, and what planner parameters should be used to terminate the cleaning task in minimum time.

7.2 Approach

This chapters considers the problem in which a robot system is given a part to clean by removing material from the surface. The overall part geometry is known to the robot, but not the material characteristics. The robot performs a cleaning *attempt* when it holds the part with a particular grasp and applies a cleaning tool to the part surface. It completes a cleaning *episode* when the part has been fully cleaned.

The approach is suited for a bimanual robot setup where one manipulator is always tasked to hold the part and the other always holds the cleaning tool. This setup implies that changing the grasp of the holding arm has a substantial time cost, as it must place the part down on a support before it can pick it up again in a new grasping position. Therefore, the primary goal of this approach is to generate plans for both arms that minimize the number of grasps needed.

7.2.1 General Problem Framework

Formulated generally, the robot runs a planner P for each attempt, which outputs a joint trajectory τ, composed of arm trajectories τ_a and τ_s for the active and supportive arms, respectively. The planner takes as input the current task instance model f_i, and a task feature vector $\mathbf{x}$, containing the object geometry

and the stained regions in the case of the cleaning task. The planner is further parametrized by a parameter vector θ, which is selected from the tool model.

$$\tau = \tau_a, \tau_s = P_\theta(\mathbf{x}, f_i) \tag{7.1}$$

The trajectory is executed by the robot, constituting a single attempt. If further attempts are required for task success, the task feature vector is reestimated and the trajectory is recomputed.

7.2.2 Part Deformation Model

The planner generates cleaning tool trajectories that are dependent on the deformation characteristics of each part that is handled, which are represented in a simplified finite element model. In this work, the system is restricted to interaction with thin, strip-like parts, which can be approximated by one-dimensional elements. However, the approach can be extended naturally to more complex models that may use elements of two or three dimensions.

7.2.2.1 Finite Element Formulation

For each part handled by the robot system, a overall linear geometry is defined, as exemplified by the part shown in Fig. 7.2. This geometry is currently given manually but is restricted to features that could reasonably be measured from a computer vision system automatically. The linear geometry is approximated by a linear mesh of N_e one-dimensional elements. A single element e_j is connected to two

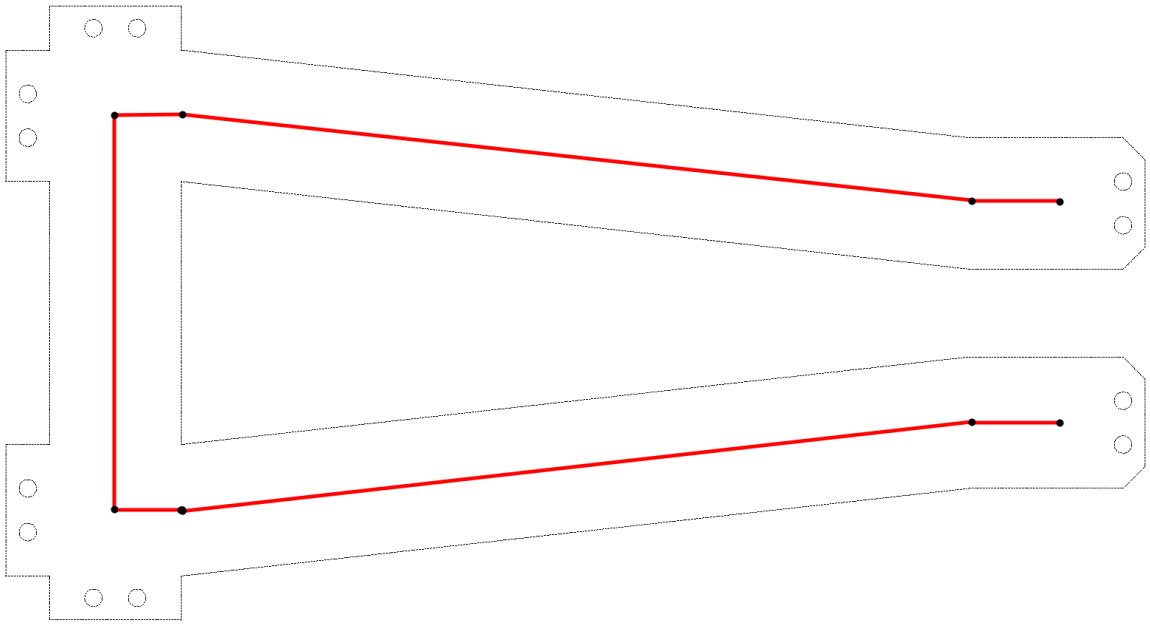

Figure 7.2: Geometry of example compliant part. The actual part outline is shown in gray, while the approximated linear geometry in shown in red.

nodes n_j and n_{j+1}, each of which has two degrees-of-freedom (DOF). Accordingly, let u_j and α_j represent the vertical displacement and rotation at n_j, respectively. The values of the $2(N_e + 1)$ DOFs for the $N_e + 1$ nodes are collected in a global displacement vector $\mathbf{V} \in \mathbb{R}^{2N_e+2}$, which fully defines the state of the mesh. Each element e_j is locally characterized by a length l_j and stiffness constant k_j, which define a local stiffness matrix K_j that relates the forces f_j, f_{j+1} and couples c_j, c_{j+1} on the element to the 2 DOF displacements at its corresponding nodes [131]:

$$\begin{bmatrix} f_j \\ c_j \\ f_{j+1} \\ c_{j+1} \end{bmatrix} = K_j \begin{bmatrix} u_j \\ \alpha_j \\ u_{j+1} \\ \alpha_{j+1} \end{bmatrix} \tag{7.2}$$

where,

$$K_j = \frac{k_j}{l_j^3}\begin{bmatrix} 12 & 6l_j & -12 & 6l_j \\ 6l_j & 4l_j^2 & -6l_j & 2l_j^2 \\ -12 & -6l_j & 12 & -6l_j \\ 6l_j & 2l_j^2 & -6l_j & 4l_j^2 \end{bmatrix} \tag{7.3}$$

The global stiffness matrix $\mathbf{K}$ is calculated by summing the contributions of the local matrices at the locations given by the indices of their corresponding nodes. The final equation to be solved is given as

$$\mathbf{KV} = \sum_{j}^{N_e} \bar{K}_j \mathbf{V} = \mathbf{F} \tag{7.4}$$

where

$$\begin{aligned} \mathbf{V} &= [u_1 \ \alpha_1 \ u_2 \ \alpha_2 \ \cdots \ u_{N_e+1} \ \alpha_{N_e+1}]^T \\ \mathbf{F} &= [f_1 \ c_1 \ f_2 \ c_2 \ \cdots \ f_{N_e+1} \ c_{N_e+1}]^T \end{aligned}$$

and $\bar{K}_j$ is a square matrix of dimension $2N_e+2$ that contains the entries of K_j inserted at the indices corresponding to the degrees-of-freedom of the j-th element.

The system of equations is made solvable by applying the known boundaries conditions of zero displacement and rotation at the node where the part is held by the gripping arm. The load vector is zero for all other nodes, with the exception of the node where the cleaning tool is applying force with a known quantity. Removing

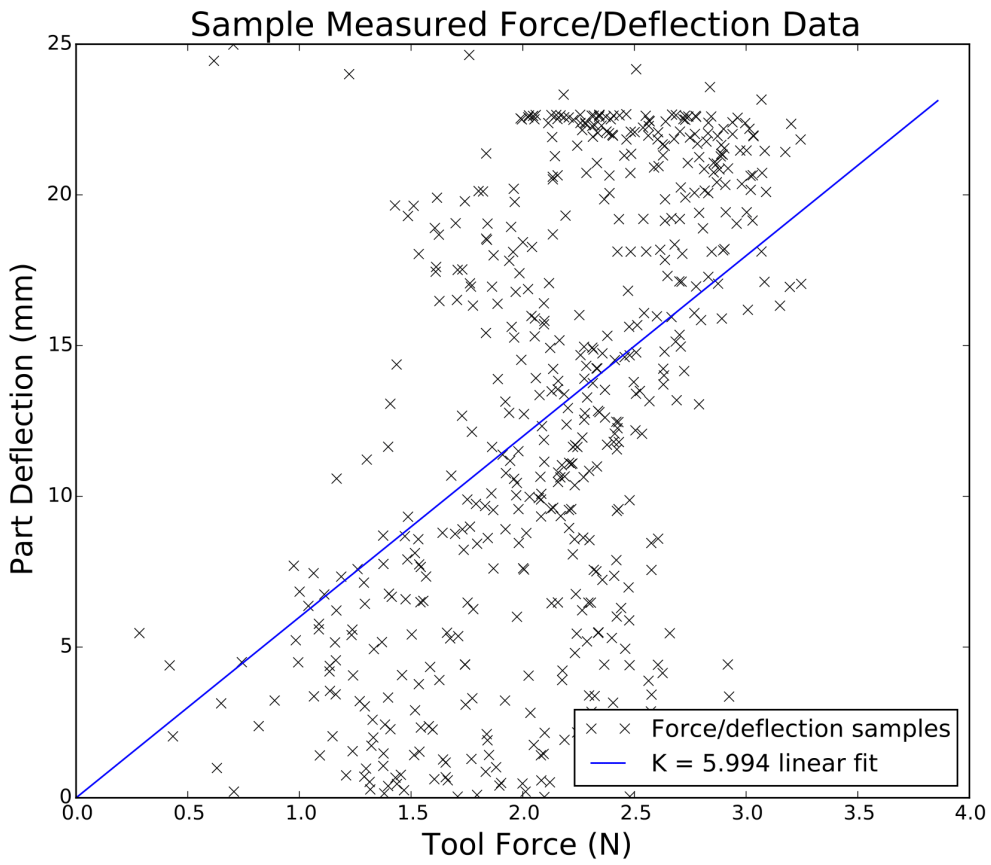

Figure 7.3: Force and deflection data measured during an example cleaning attempt. The data contains substantial noise but a linear fit to obtain a stiffness estimate is still possible.

the equations for the node that is held at zero displacement then gives a system of $2N_e$ linear equations with $2N_e$ unknowns that can be solved.

7.2.3 Model Improvement

During a single cleaning attempt, one robot holds the part at a fixed location and another robot applies a normal cleaning force at other locations on the part. The robot is able to report at high frequency the current position and force experienced

by the tool tip during the cleaning motion. Figure 7.3 shows an example of data collected during such an attempt. This set of samples can be used to estimate an effective stiffness of the part between the grasping and force application points. This relationship is expected to be linear, though as can be seen from the data, the measured values contain substantial noise. Nevertheless, a linear fit of the data points does give a physically reasonable value for the part stiffness.

7.2.3.1 Direct Stiffness Parameter Scaling

The first model improvement method described is to compare measured effective stiffness values directly to the expected value predicted by the current part deformation model. A global stiffness matrix is calculated in the same way as in the previous section, with the same boundary conditions applied for the two nodes. The effective stiffness between the two nodes can then be calculated as the entry in $\mathbf{K}^{-1}$ that relates the force to the displacement at the free node. The stiffness constants of the current mesh model are then uniformly scaled by the difference between the calculated and measured stiffnesses so that the model behavior matches the measured value. Since each mesh element contributes linearly to the overall deformation behavior, each element can be scaled by the same factor computed for the effective stiffness to achieve it. This provides a local update to the part deformation model after each cleaning attempt. As the robot is tasked to clean stains in new regions of the part and collects more data, it eventually gains a full model of the true deformation behavior as the local updates converge to the true model. Long before the

full true model is acquired, however, the knowledge gained is typically sufficient to plan correct grasps. This is a natural consequence of the grasp planner attempting to minimize the distance between the support and force application points, which implies the updates after measuring the part behavior will be acutely targeted.

7.2.3.2 Batch Stiffness Parameter Estimation

The main limitation of the previous method is that updates can only occur after individual measurements of effective stiffnesses, and the past data points are not used in each subsequent update. The alternatively is to use a batch update method on all of the collected stiffness data.

Over the course of one or more task attempts, samples are generated from many different force application points and one or more grasping points. This produces a set of measured effective stiffness values relevant to some subset of the elements in the mesh approximation, $\mathbf{k}_m \in \mathbb{R}^M$, with corresponding coordinates of the grasping and loading points, $\mathbf{x}_g \in \mathbb{R}^M$ and $\mathbf{x}_l \in \mathbb{R}^M$, where M is the number of unique pairs of grasping and loading points.

At the end of each attempt, the collected data can be provided to the update algorithm (Algorithm 7) which will adjust the stiffness parameters of the mesh model to minimize the error computed from the measured effective stiffness values. The update algorithm is structurally the same process as backpropagation learning for neural networks, where the gradient of an error function is used to drive local

updates to the model parameters. We first define the overall error function which depends on the model parameters κ_j and the measured data:

$$E(f_p, \mathbf{k}_m, \mathbf{x}_g, \mathbf{x}_l) = \frac{1}{M} \sum_{i=1}^{M} (k_p^{(i)} - k_m^{(i)})^2 \tag{7.5}$$

The predicted effective stiffness of the i'th measurement, $k_p^{(i)}$, is determined by solving the overall finite element equation (Eq. 7.4) with the boundary condition imposed by the corresponding grasping point $x_g^{(i)}$, and a unit force applied at the loading point $x_l^{(i)}$. The resulting effective stiffness value is then simply the inverse of the displacement computed for the node at the loading point.

The derivative of E can be easily computed analytically except for the partial derivative of the calculated effective stiffness with respect to each stiffness parameter of the model, because it involves a matrix inversion. We calculate this partial derivative with a finite difference approximation, adjusting the stiffness parameter by a small percentage and measuring the change in the resulting effective stiffness. This allows the full gradient of the error function to be calculated:

$$\nabla E_j = \frac{2}{M} \sum_{i=1}^{M} (k_p^{(i)} - k_m^{(i)}) \frac{\partial k_p^{(i)}}{\partial \kappa_j}, 1 \leq j \leq N_e \tag{7.6}$$

We then use an off-the-shelf gradient based optimizer to modify the stiffness parameters in f_p using Eqs. 7.5 and 7.6. This produces the next estimate for the κ_j parameters that will be used in for the next cleaning attempt.

There are a few implementation details that can be used to improve the optimization speed for this particular problem. First, as in neural network backpropaga-

tion, this algorithm is susceptible to very small gradient values when the model has many parameters. Commonly used units for stiffness parameters lead to relatively large parameter values and so some normalization is important when the optimizer is running. Second, when comparing computed effective stiffness values between a close pair of points versus a distant pair, there can be several orders of magnitude of difference. This leads to some samples having dramatically larger effect on the direction of the gradient. Therefore, we also normalize the computed and measured effective stiffnesses by the distance between the grasping and loading points, which improves the overall learning ability.

Algorithm 7 Stiffness Model Update

Inputs: Measured effective stiffness values ($\mathbf{k}_m$), Current part model ($f_p = \{\kappa_j\}$), Grasp coordinates ($\mathbf{x}_g = \{x_g^{(i)}\} \subset \mathbb{R}$), Load coordinates ($\mathbf{x}_l = \{x_l^{(i)}\} \subset \mathbb{R}$).
Outputs: Updated part model (f_p').

$f_p' \leftarrow \{\kappa_j\}$
$M \leftarrow |\mathbf{k}_m|$
while f_p' not converged **do**
    $\mathbf{k}_{pred} \leftarrow$ EffectiveK(f_p', $\mathbf{x}_g$, $\mathbf{x}_l$)
    for $j \leftarrow 1$ to N_e **do**
        for $i \leftarrow 1$ to M **do**
            $\frac{\partial k_p^{(i)}}{\partial \kappa_j} \leftarrow$ FiniteDiffEst(f_p', j, $x_g^{(i)}$, $x_l^{(i)}$)
        end for
        Compute E (Eq. 7.5)
        Compute ∇E (Eq. 7.6)
    end for
    $\{\kappa_j'\} \leftarrow$ OptimizerStep(f_p', E, ∇E)
    $f_p' \leftarrow \{\kappa_j'\}$
end while

7.2.4 Cleaning Arm Planner

In this work, the active arm holds a cleaning tool and is tasked to pass it over a set of designated stain regions while applying a normal force against the part surface. A simple coverage planner is used that moves the cleaning tool in a set of parallel linear passes to cover the stained regions identified by the perception algorithm. The tool passes are set parallel to the y-axis of the part, and the distinct regions provided by the perception algorithm are planned separately. While simple and suboptimal, this method is useful for illustrating our general learning approach. In the general case, our approach will also work effectively with any coverage planning algorithm that can be parameterized by a small set of variables.

The planner used here was parameterized with two values: the normal force applied by the tool to the part surface and the spacing between the tool passes over the regions detected as stains. These parameters provided complementary influences on the cleaning behavior. Increasing the tool force typically leads to better cleaning performance as the tool will remove more material in each pass. However, it also increases the amount of part deformation, implying more changes in grasp locations will be needed to provide support closer to the force application points. In contrast, increasing the tool pass spacing will lead to faster cleaning time, but inferior cleaning performance.

Algorithm 8 Cleaning Task Attempt Planner

Inputs: Task parameters ($\mathbf{x} \in \mathbb{R}^n$), Current part model ($f_i$), Task specification (stained regions) ($\sigma \subset \mathbb{R}^2$), Possible grasp parameters ($G \subset \mathbb{R}$)

Outputs: Number of distinct grasps (N), Cleaning tool trajectory ($\tau_a = \{\tau_a^{(i)}, 1 \leq i \leq N\}$), Grasp location parameters ($\{g_1, g_2, \ldots, g_N\}$)

function PLAN($\mathbf{x}$, f_i, σ, G)
  $\mathcal{P} \leftarrow \varnothing$
  $\tau_a \leftarrow P_{\mathbf{x}}(\sigma)$ (parameterized coverage planner)
  $N \leftarrow 1$
  while $N < N_{max}$ **do**
    $c_f(\mathbf{g}) \triangleq$ GRASPCOST($f_i, \tau_a, \mathbf{x}, \mathbf{g}$)
    $\mathbf{g}^* \leftarrow optimize(c_f(\mathbf{g}))$ for $\mathbf{g} \in G^N$
    $d_{max} \leftarrow max(\{f_i(\tau_a^{(i)}, \mathbf{x}, \mathbf{g}) \; \forall \; \tau_a^{(i)} \in \tau_a\})$
    if $d_{max} < d_{thresh}$ **then**
      $\mathcal{P} \leftarrow \tau_a, \mathbf{g}^*$
    else
      $N \leftarrow N + 1$
    end if
  end while
  return $\mathcal{P}$
end function

function GRASPCOST(f_i, τ, $\mathbf{x}$, $\mathbf{g}$)
  $D_{pred} \leftarrow \{f_i(\tau^{(i)}, \mathbf{x}, \mathbf{g})$: $\tau^{(i)} \in \tau\}$
  $\hat{d}_{avg} \leftarrow mean(D_{pred})$
  return $\hat{d}_{avg}$
end function

7.2.5 Grasping Planner

After the tool path is generated to cover the stained regions, the coordinating actions of the supportive arm must be planned. In this case, the arm must grasp the part at one location or change between multiple grasp locations along its perimeter to prevent excessive deformation from occuring during cleaning. For simplicity, a finite set of possible grasp locations G is predefined for each part, defined as real-valued distances along the part perimeter. The robot then searches for the optimal subset of G that will enable cleaning of the part with the known force without exceeding a deformation threshold (Algorithm 8). Essentially, the algorithm searches through all possible grasp locations, starting from a single grasp and incrementally adding multiple grasps if none can be found with a predicted deformation below a specified threshold. When evaluating the predicted deformation of the points on the planned tool path, the part model deformation is computed between the mesh nodes corresponding to the force application point and the particular grasp closest to it. Importantly, the closest grasp is not determined by Euclidean distance but distance along the mesh.

In deciding the grasp point(s), the approach here uses a cost function that considers any deformation beyond the threshold to be unacceptable, and the optimizer will continue to add new grasp points to the plan, dramatically increasing the needed attempt time, in order to reduce the maximum deformation below the threshold. Alternatively, a cost function cost be used without an explicit threshold with the goal of more directly balancing the time cost with the expected deforma-

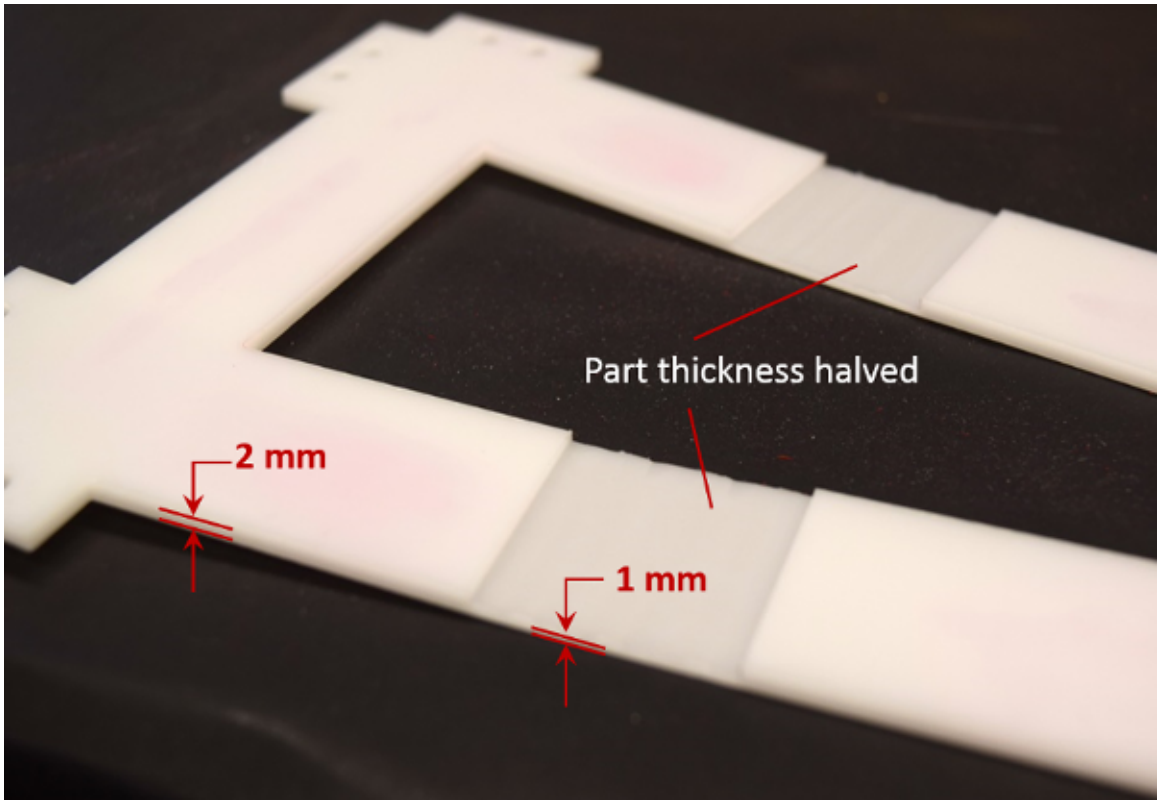

Figure 7.4: Thickness distribution of the part

tion. In particular, the cost function described in Chapter 5 (Eq. 5.2) with weights learned by human demonstrators can naturally be used by the optimizer, allowing the system to directly adopt some aspect of a human expert's strategy.

This algorithm minimizes both the number of grasps used during the cleaning attempt as well as the average deformation during the attempt. Note that a tradeoff may be possible in that additional grasps can lead to even lower average deformation, which may be desireable in the case of extremely delicate parts, for example. In this case, however, the algorithm is constrained to always return the smallest number of grasps possible.

7.2.6 Tool Performance Model

For each cleaning attempt, the trajectory generated by the planner is substantially affected by the tool parameters used. This effect is not limited to the path on the part surface that the cleaning tool follows, but includes where the grasping arm should hold the part since the parameters may include features that will impact the amount of deformation, such as the normal force. Before each attempt, the system selects the parameters that it expects will lead to the fastest cleaning time. Note that in general this involves optimizing some trade-off since more effective cleaning parameters will typically lead to longer cleaning times. After each attempt, the actual cleaning performance is measured and the model is updated to improve future parameter selection. The tool model can be shared and improved by all episodes that involve the same tool and part material. This enables the robot system to do long-term learning of the cleaning model.

The tool model f_t, predicts cleaning performance μ for a particular parameter vector θ, i.e., $\mu = f_t(\theta)$. The cleaning performance metric used is the expected area of the stain that will be removed, which is measured empirically by a computer vision system. An initial image is taken of the stained regions and the pixels are labeled by color using a k-means classifier with three clusters as clean, dirty or background. After the cleaning attempt, the fraction of pixels forming part of the stain which are reclassified from dirty to clean is reported as the cleaning performance.

To model the tool performance so that the value of new planner parameters can be predicted, a Gaussian Process Regression [76] model with a linear mean

Algorithm 9 General Task Learning

```
function TaskEpisode(x, f_i, σ)
    done ← false
    while NOT done do
        P ← Plan(x, f_i, σ)
        X_p ← execute(P)
        σ_n ← Remeasure task specification
        f_i ← modelupdate(f_i, X_p)
        σ ← σ_n
        if σ = ∅ then
            done ← true
        end if
    end while
end function

f_i ← init task instance model
while true do
    x ← Select planner parameters
    C_d ← measure dirty regions
    TaskEpisode(x, f_i, C_d)
end while
```

function is used. A Gaussian process in $\mathbb{R}^n$ is fully defined by a mean function $m : \mathbb{R}^n \to \mathbb{R}$, and a covariance function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, generally the squared exponential $K(x_1, x_2) = exp(\frac{-||x_1 - x_2||}{l^2})$. The length-scale parameter l controls how strongly the data samples influence the model prediction of their neighboring points and is tuned using cross-validation or expectation maximization to best fit a batch of data. This then fully defines the tool performance model $f_t(\theta) \sim \mathcal{N}(m(\theta), K)$.

In this work, we specify $m(\theta) = w^T\theta$, with $w \in \mathbb{R}^n$. The weight vector w is estimated from the current data samples and enables the model to extrapolate the performance behavior beyond the range where data samples have been taken.

When selecting the next parameters to use, the possible parameter values are taken from a discrete grid over the acceptable range of values. This allows the

optimizer to quickly find the next best parameters rather than doing a search over the full space.

7.3 Results

7.3.1 Part Finite Element Model

Initial results of the part model learning algorithm were obtained first on a simulation of a linear mesh. To test the direct parameter scaling method, a meter long mesh was used, divided into 100 elements, with a continuously varying stiffness from 50 to 250 Nm^2, seen in Fig. 7.5. An initial model of the mesh is initialized with a constant stiffness value of 150 Nm^2. For each iteration, a pair of random points along the mesh are selected and the effective stiffness value between them is queried on the true mesh. The estimated mesh is then updated over the corresponding elements using the scaling algorithm. Fig. 7.5 shows the stiffness estimates after 5, 10, 15, 20, and 1000 iterations of this procedure. Figure 7.6 shows the average stiffness estimate error of each element up to 100 iterations. It can be seen that while the convergence is relatively slow, the mesh does roughly acquire the same overall behavior as the true mesh within a small number of attempts. Furthermore, the average error curve indicates an overall downward trend with very few increases over subsequent iterations. The same trend continues up to 1000 iterations. This provides some assurance that the algorithm is stable and will not lead to diverging stiffness estimates. For this example with randomly generated test points, we did

observe that the average error drops close to zero ($\sim 5Nm^2$), but only after close to 1000 iterations.

The main artifact noticeable from the parameter scaling method is the amount of noise after many iterations, which is due to the algorithm not taking into account the measurements other that the most recent one when performing updates. This is quite different from the results produced from the batch update algorithm. For this method, a 30 cm long mesh was tested, divided into 30 elements, with a continously varying stiffness from 50 to 250 Nm^2, seen in Fig. 7.7, with the initial model of the mesh initialized to a constant value of 100 Nm^2. A full set of data is then used to update the mesh, with each data point consisting of a sample of the effective stiffness value between two randomly selected points on the mesh. The estimated mesh is then updated over the corresponding elements using the learning algorithm. Fig. 7.7 shows the stiffness estimates learned when using 5, 10, 20, and 100 samples to perform the estimate, compared to the true parameter values. Figure 7.8 shows the average relative error of the stiffness parameters of the updated mesh after several experiments with varying numbers of stiffness samples. It can be seen that the convergence is quite fast, with the mesh roughly matching the as the true mesh within a small number of attempts, especially in the center area. The inaccuracies towards the end points of the mesh are largely a function of there being fewer samples that include those elements, due to the random sampling process. Note that after 100 samples are taken, only a few elements at either end have substantially incorrect estimates, while the elements in the center match the true values almost perfectly. Further experiments were conducted out to 1000 sampled stiffnesses and

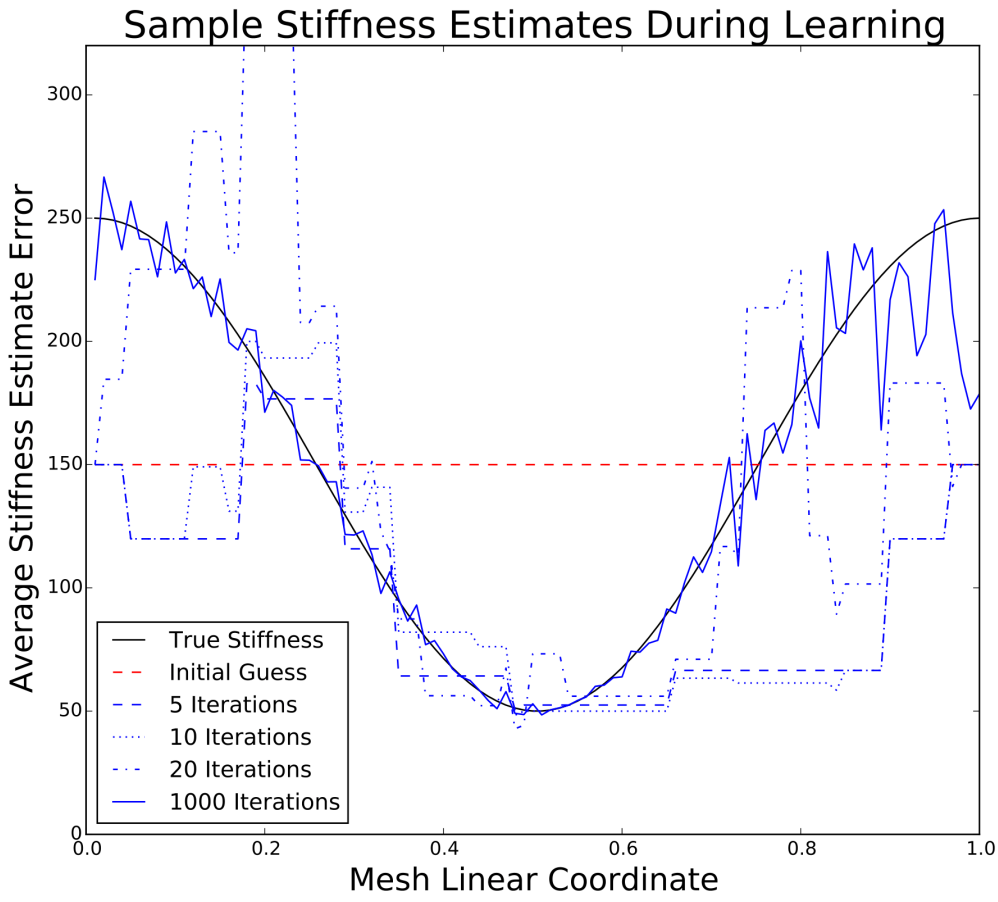

Figure 7.5: Element stiffness estimates after random tests and updates using the stiffness parameter scaling method.

the convergence is maintained from Fig. 7.8, stabilizing around 3% error at 1000 samples. Due to the superior accuracy and convergence of the batch estimation method, the remainder of the results are shown using only this algorithm.

7.3.2 Physical Part Cleaning

We investigated the model parameter estimation for an example planar part made from thin plastic, corresponding to the design shown in Fig. 7.2. The part was

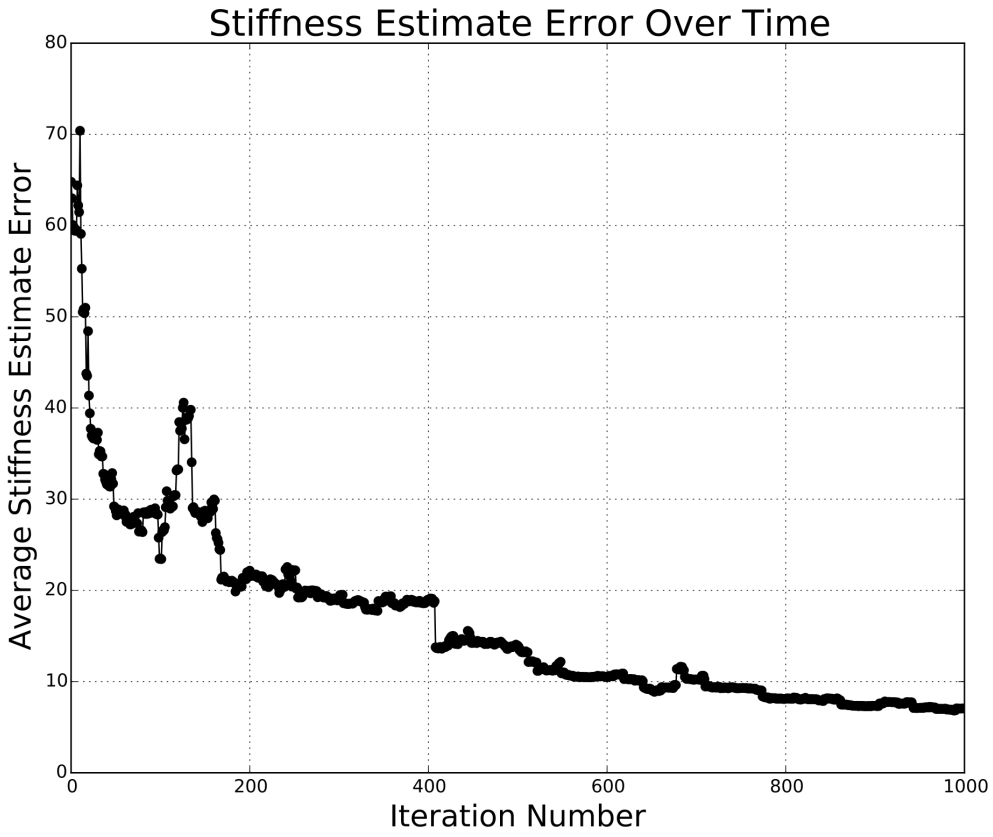

Figure 7.6: Average element stiffness estimate error using single-measurement parameter scaling as a function of the number of iterations up to 100 iterations.

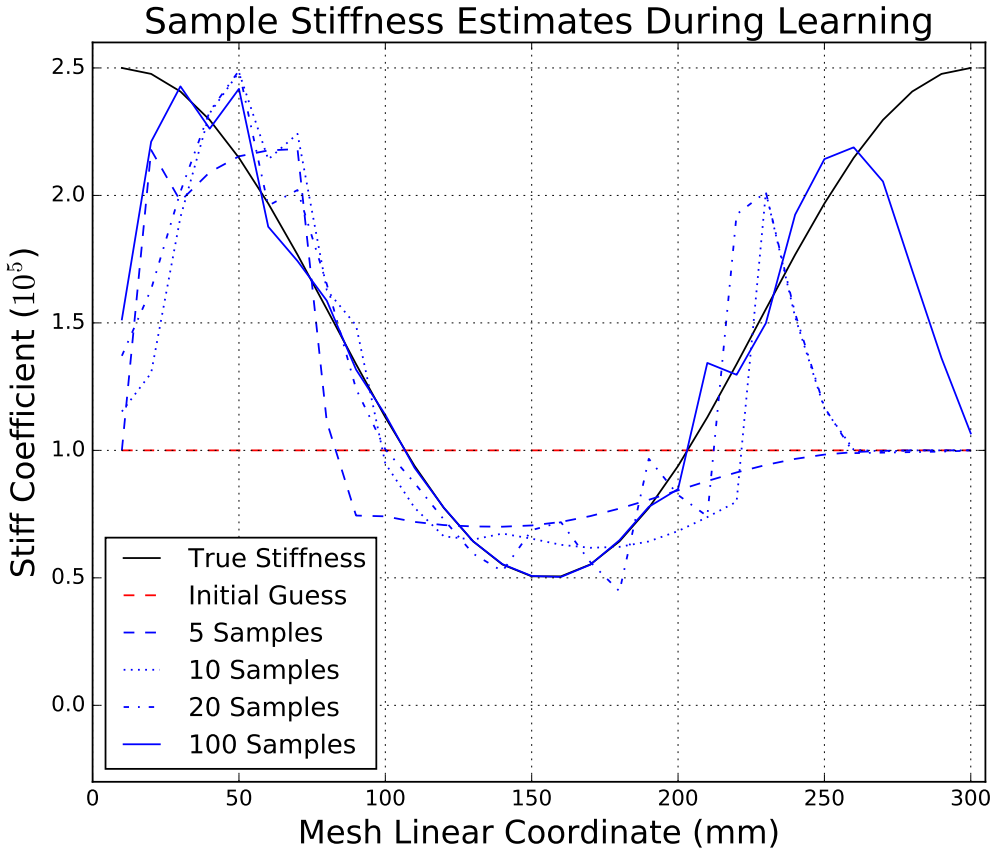

Figure 7.7: Element stiffness parameters estimated using the batch method on a variable number of samples at random locations on the mesh.

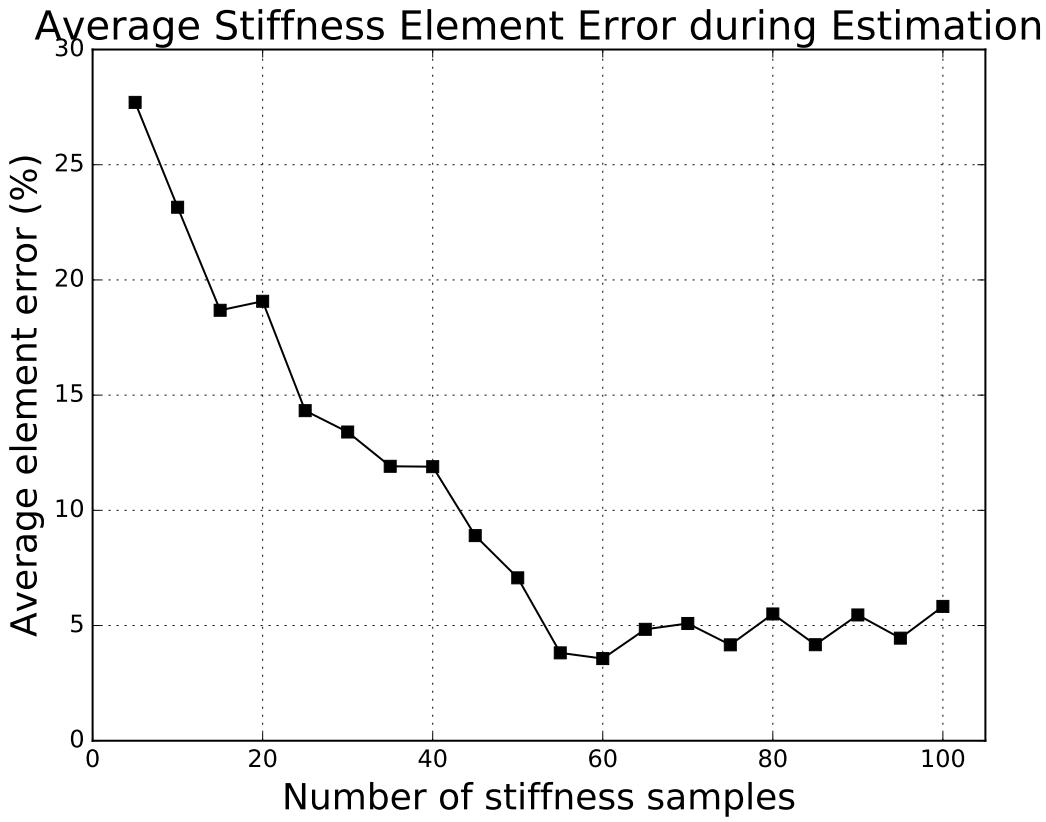

Figure 7.8: Average relative error of the element stiffness parameters as a function of number of samples used in the batch estimation for the model update.

of uniform stiffness through its length, except at two segments along the arms where the thickness was halved as shown in Fig. 7.4. The applied strains were created with a highly visible marker and were calibrated to require a substantial amount of force to remove. During human experimentation with the cleaning, it was always required to grasp the part close to the stain in order to apply any significant force, and cleaning while grasping across the extra-compliant segment was generally not possible. The maximum deflection threshold specified was 20 mm.

The part deformation model estimation is demonstrated with a cleaning episode shown in Fig. 7.9. The model is initialized with a uniform stiffness value, and so the planner has no knowledge of the low stiffness regions. Therefore, for the first attempt (Fig. 7.9(a)), the planner selects only a single grasp point while covering all three stained regions. Initial cleaning is successful for the regions to the left, but the region in the lower right is across from the low stiffness area, and so the deflection threshold is quickly surpassed, and the attempt is halted. The robot is still able to provide sufficient force/deflection samples before the attempt stops, and the measured stiffness data allows the update algorithm to immediately lower the stiffness parameter estimates in the vicinity of the lower stiffness region (see Fig. 7.11). For the second attempt (Fig. 7.9(b)), the planner realizes that two grasps are required, but it selects one across the upper region of low stiffness. This also results in the attempt quickly terminating, but is sufficient to identify the other low stiffness region in the deformation model. The second grasp from attempt 2 ends successfully, however, and fully cleans the lower right stain. Attempts 3 and 4 finish

cleaning the other two stains, with the planner aware that the only valid grasping region is on the correct side of the low stiffness regions.

The final estimated stiffness parameters of the part model after the full episode are shown in Fig. 7.11. Note that the low stiffness regions were distinctly identified in the correct locations, and also that the areas with higher stiffness were raised to match their value. On the left side, the model stays at the initial stiffness estimate simply because no data was collected with a grasp or stain in that region. As more episodes are conducted with different stained regions, we would expect that the model would closely approximate the true parameters in all areas. Furthermore, even though the learned model after one episode has quite a bit of estimation noise, it is already sufficiently accurate to generate correct grasping plans for other stain configurations. A second cleaning episode was run with stains on both ends of the part arms and the center vertical strip (Fig. 7.10). The planner was immediately aware that three grasping points would be required, and only two attempts were needed to fully clean the part.

Overall, the results show fast cleaning performance even when initialized with limited knowledge of the part deformation behavior, and more rapid performance when a previously learned model is available. While the system cannot learn a fully accurate stiffness model in only a few cleaning attempts, it was shown to be sufficient for the planner to select correct grasp points for effective cleaning.

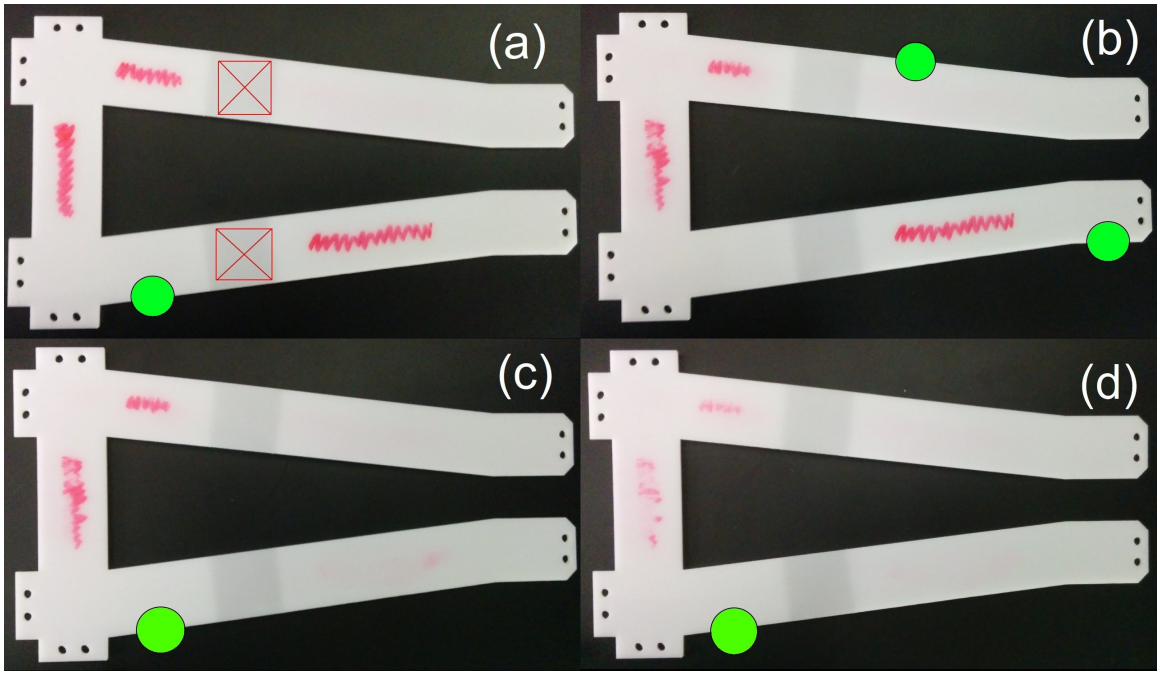

Figure 7.9: Example first cleaning episode of part with variable stiffnesses. The regions of low stiffness are marked in red in (a). The grasp points used for each attempt are marked with green circles.

7.4 Summary

This chapter demonstrates an approach for a bimanual cleaning system that is capable of online learning of simple part models using a simple finite-element model. Using a simple incremental model update, the system was able to learn rapidly enough to make correct grasp and improved parameter selection after just a few iterations of attempts and updates. By using a finite-element model as the core of the task dynamics model, the robot is able to more effectively learn the dynamics of non-uniform tasks, specifically parts with varying stiffnesses. The primary contribution is the updating algorithm which enables the system to converge towards the true parameters of the task model faster than the black-box input/output model.

Figure 7.10: Second cleaning episode stain configuration and cleaned part after two attempts.

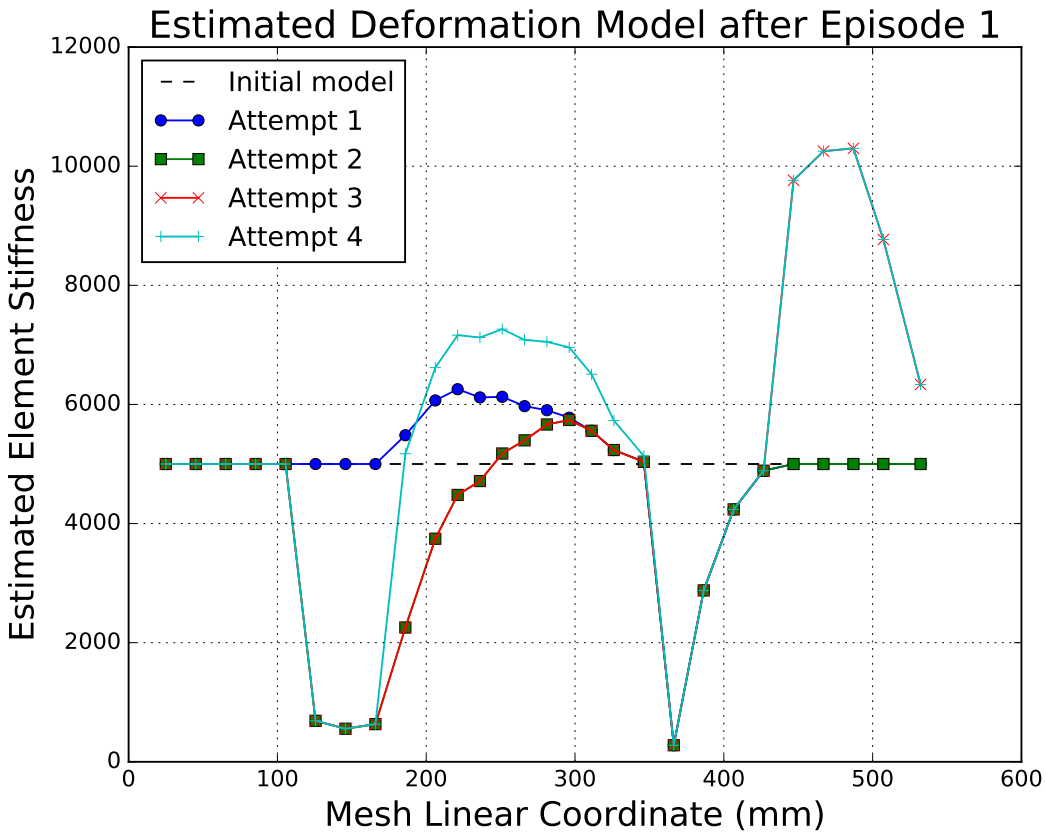

Figure 7.11: The estimated part model after using the data collected during the cleaning attempts. Each attempt only affects the stiffness parameters of a subset of the mesh, so many of the estimates are unchanged and overlapping. The centers of the stiffness regions on the part correspond to mesh coordinates of 140mm and 390mm.

Chapter 8: Conclusions

This chapter summarizes the contributions of this dissertation and the anticipated benefits to the larger community. Future research directions to continue addressing the presented issues and questions are also discussed.

8.1 Intellectual Contributions

The main intellectual contributions of this dissertation include the following:

- *Use of human demonstrations for parameterization of tasks involving nonrigid materials*: This first contribution is how initial task models can be created by making use of human demonstrations on both the dynamic pouring task and the compliant object cleaning task. In the case of the cleaning task, the demonstrations were used to inform the task cost function used by a trajectory planner, enabling the potential of learning implicit strategies or priorities from demonstrations of other tasks. The initial models learned from demonstrations can then be improved on by the robot as it attempts the tasks, and gains additional experience.

- *Integration of learning methods with known task structure*: The tasks researched in this dissertation and many others, can be accomplished with a combination of planning and learning methods. This allows a system to be effective, as a programmer can specify the elements of the task that are easily described explicitly to the robot, and the more complex elements involving nonrigid object dynamics can be learned. For the pouring task, this is demonstrated with a system that learns parameters used by a classical planner to generate the manipulator trajectory. For the cleaning task, the planner optimizes based on the current model of the part dynamics, which is unknown to the programmer. Furthermore, additional structure is provided to the part dynamics model in the form of a finite-element model, which enables faster learning of non-uniform part dynamics.

- *Study of function approximation algorithms used in task models*: This dissertation explores two primary methods of function approximation for the task models: linear models and Gaussian processes. The two methods are used and discussed in different approaches which make use of their advantages and minimize their disadvantages. The results are informative for a system designer considering the method to be used for a new task.

- *Strategy for rapidly finding solutions to new task variations*: The main goal of this dissertation is to demonstrate algorithms that can learn to successfully perform new variations of tasks with a minimum number of attempts. For the pouring task, the presented approach was able to find solutions for new

pouring volumes quickly in the space of the pouring policy parameters, using single parameter exploitation updates with a set of local linear models. For the cleaning task, the robot could quickly learn a deformation model to accurately optimize grasping locations by incorporating data acquired for task attempts into the model of the manipulated object. The presented algorithms are general in nature and can potentially be applied to a variety of different tasks where the use of robots might be desired.

8.2 Anticipated Benefits

This dissertation presents methods for automated learning of manipulation tasks involving nonrigid materials. It is anticipated that the research should be applicable to a wide variety of industrial scenarios and may be of use in developing approaches to enable robotic execution of new tasks. Having the robot learn new tasks automatically without manual programming will have significant cost savings. Once the learning framework is established, the robot can gain knowledge of a new task much faster than it would take an expert to program through its own experience attempting the task or through demonstrations by a human. Having the ability to learn tasks involving nonrigid objects such as fluids and flexible materials will also allow the robot to do much more than is currently possible where such tasks are often impossible to manually program and infeasible to compute with a traditional planner.

8.3 Potential Future Directions

There are several potential avenues for further investigation of the issues explored in this dissertation. A few are discussed here in relation to the particular approaches developed in the preceeding technical chapter:

- *Joint use of imitation learning approaches*: For the imitation learning approaches from Chapters 3 and 5, future research can explore the question of how to two presented approaches can be incorporated into the same overall learning framework rather than relying on one only. As they address different aspects of performing the task (trajectories vs. strategy for the planner), they can plausibly be integrated into a single system. A task that could be learned in such a system would have a breakdown of components or state variables where some would driven by a planner influenced by the learned cost parameters, while the others would recieve direct trajectory inputs generalized from demonstrations. The overall system would require analysis on the sensitivity of each subsystem to the generalization of the other, and where potential conflicts might arise, especially if both subsystems are trained using data from the same demonstration set. Through careful separation of the aspects controlled by each subsystem, however, it should be possible to have them complement each other to acheive improved overall learning performance.

- *Additional flexibility in learned trajectory dynamics model*: The approach presented in Chapter 4 has several possibilities for extensions. How the complexity

and computational limits of the algorithm scale to higher dimensions is still an open question. The current approach updates all local models as new data is acquired but the formulation naturally extends to an incremental learning scheme where only the models in the vicinity of the newly sampled data point are adjusted. For long-term learning, this should lead to an asymptotically constant update time as the neighborhoods of the current data points shrink in densely sampled regions, suggesting that new data will not influence an ever-growing number of local models.

Additionally, the exploitation strategy used was validated primarily on local linear models but was also shown to be possible with other modeling algorithms. This suggests the approach could be expanded to become model-agnostic and even use a hybrid approach where different regions of the parameter space would be modeled by different algorithms, depending on the function behavior. The linear models may be best suited to regions of sparse data providing high computational performance and reasonable extrapolation ability. A Gaussian process may be more usefule to provide more accurate predictions in regions of high variability, and LWPR could be used when sufficient samples had been acquired such that the other algorithms are slowed down enough that computational time becomes the driving component of the cost. These improvements may enable the approach to be competitive for higher dimensional problems as well.

- *Generalizability of deformation model*: For Chapters 6-7, the major avenue of future investigation is in more generalizable forms of the deformation model representing the task dynamics. The two approaches showed data-driven updates of both a black-box deformation prediction model, and a known-structure finite-element model. Analogous to the hybrid modeling mentioned for the trajectory dynamics model, a natural question is the possibility of using both techniques on the same part model. This may take the form of using a more accurate Gaussian process model on certain areas of the part geometries according to some heuristic, or layering the black-box model over the finite-element model, which can then be much coarser.

  There is also the question of extending the update algorithm for the part model to work on parts with 2D and 3D geometry characteristics. When new data is recieved, the update algorithm must decide which elements are relevant to be changed and which should be expected to stay the same. One approach may involve a precomputed database based on initial assumed model of how different regions affect the overall task behavior and weight them accordingly. Online decisions can be made by the planner and update algorithm using the precomputed data, and the database can be updated as determined by an accuracy heuristic or just in the downtime between the trials of different task instances.

  Finally, there is the question of how the trajectories generated through the planner from the current model is coupled to the learning the parameters of

the tool model to accurately predict task effectiveness. With an inaccurate tool performance model, the trajectory found by the planner may be highly ineffective and the heuristics used to adjust the parameter here may require more task trials than necessary. Here the question to investigate is whether there is an optimal strategy between using parameters that gain more information on the tool model but may require more task execution time due to lower efficacy. One strategy may be primarily emphasize learning the part dynamics model first before engaging in tool parameter exploration. Any tradeoff here should also be sure to address the uncertainty that is learned in the part model and make decisions to minimize the expected overall time in the long-term.

Appendix A: Gaussian Process Regression Method

GPR fits a Gaussian process over a set of training samples comprised of multi-dimensional inputs and real-valued outputs, and produces a predicted normal distribution for the output of any point in the input space. While a Gaussian process can be thought of as an infinite-dimensional normal distribution, when making predictions for new input points the formulation depends only on the training data, which makes the algorithm simple to implement. The standard training and prediction algorithm is derived in [76] and gives efficient prediction for specified points in the input space. However, it does not provide any guidance on how to select test points. Here we extend the basic algorithm to be applicable for our use case, where we are searching the input space for a point that corresponds to a desired output.

The standard algorithm takes a data set of previous trials, with inputs $\mathcal{X} \subset \mathbb{R}^n$, and outputs $\mathcal{Y} \subset \mathbb{R}$, and returns a predicted mean (μ_*) and variance (σ_*) of the output for a test point, $x^{(*)}$. The most important component of the GP is the covariance function, $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, which controls the similarity of neighboring points in the input space and enforces the smoothness of functions that are fit to

the training data. We use the standard squared exponential covariance function of the form

$$k(\mathbf{x}, \mathbf{y}) = \sigma_f e^{(\mathbf{x}-\mathbf{y})^T D(\mathbf{x}-\mathbf{y})}. \tag{A.1}$$

Here D is a symmetric, positive-definite matrix that defines a distance metric. In our experiments we used a diagonal matrix with each diagonal element as a free parameter that controls how quickly the influence of two points drops as the distance between them increases along a particular axis. The other free parameter used is σ_f, which is a uniform scaling factor for the covariance.

The covariance function is used to compute correlations between the training set and the test point. The matrix $K \in \mathbb{R}^{m \times m}$ is the covariance matrix of the training data, computed by $K_{ij} = k(x^{(i)}, x^{(j)})$ for all m points in $\mathcal{X}$. The covariance vector k_* is computed by pairing $x^{(*)}$ with each $x^{(i)}$ from the training data. The predicted mean and variance of the output corresponding to $x^{(*)}$ are then given by:

$$\begin{aligned} \mu_* &= k_*^T (K + \sigma_n^2 I)^{-1} \mathcal{Y} \\ \sigma_* &= k(x^{(*)}, x^{(*)}) - k_*^T (K + \sigma_n^2 I)^{-1} k_*. \end{aligned} \tag{A.2}$$

Notice that the majority of the computation is inverting the matrix $K_n = K + \sigma_n^2 I$ which can be quite expensive if its dimensions are large. However, as K_n is only dependent on the training data, it can be computed once and cached for future

predictions of test points. This is crucial for efficient use of the prediction in our extension.

After a training set of data is obtained, prediction accuracy can be increased by finding a set of hyperparameters that maximizes the likelihood of the training data. In our case, the hyperparameters consist of the diagonal elements of D and σ_f, forming the parameter vector θ. The log-likelihood of the hyperparameters for a given training set can be analytically derived as

$$logp(\mathbf{y}|X,\theta) = -\frac{1}{2}\mathbf{y}^T K^{-1}\mathbf{y} - \frac{1}{2}log|K| - \frac{n}{2}log2\pi. \tag{A.3}$$

To maximize this function, a standard gradient ascent algorithm in the space of the parameter vector θ is used as the derivative can be analytically computed. For each parameter θ_i, this derivative is:

$$\frac{\partial}{\partial\theta_i}logp(\mathbf{y}|X,\theta) = \frac{1}{2}tr[(K^{-1}\mathbf{y}(K^{-1}\mathbf{y})^T - K^{-1})\frac{\partial K}{\partial\theta_i}]. \tag{A.4}$$

Note that the calculation of K (and therefore K^{-1}) is dependent on the values of θ, essentially requiring a full retraining of the GP at each new point in the hyperparameter space. This cost is sufficient that it becomes prohibitive to reoptimize the hyperparameters every time new data is acquired, for example. However, we found it is possible to obtain sufficient prediction accuracy when finding the optimal parameters for the initial training set only.

Bibliography

[1] S. Miller, J. van den Berg, M. Fritz, T. Darrell, K. Goldberg, and P. Abbeel. A geometric approach to robotic laundry folding. *The International Journal of Robotics Research*, 31(2):249–267, 2012.

[2] M Matsushima, T Hashimoto, M Takeuchi, and F Miyazaki. A Learning Approach to Robotic Table Tennis. *Robotics, IEEE Transactions on*, 21(4):767–771, aug 2005.

[3] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA urban challenge: Autonomous vehicles in city traffic*, volume 56. springer, 2009.

[4] Mozafar Saadat and Ping Nan. Industrial applications of automatic manipulation of flexible materials. *Industrial Robot: An International Journal*, 29:434–442, 2002.

[5] Torgny Brogårdh. Present and future robot control development–An industrial perspective. *Annual Reviews in Control*, 31(1):69–79, 2007.

[6] R. C. Miall. Connecting mirror neurons and forward models. *Neuroreport*, 14(17):2135–2137, 2003.

[7] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[8] Lu Yingwei, N Sundararajan, and P Saratchandran. A Sequential Learning Scheme for Function Approximation Using Minimal Radial Basis Function Neural Networks. *Neural Computation*, 9(2):461–478, 1997.

[9] George Konidaris, Sarah Osentoski, and Philip Thomas. Value Function Approximation in Reinforcement Learning using the Fourier Basis. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*, pages 380–385, 2011.

[10] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[11] R J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[12] Jens Kober, Andreas Wilhelm, Erhan Oztop, and Jan Peters. Reinforcement learning to adjust parametrized motor primitives to new situations. *Autonomous Robots*, 33:361–379, 2012.

[13] A J Ijspeert, J Nakanishi, and S Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 2, pages 1398–1403, 2002.

[14] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors. *Neural computation*, 25(2):328–373, 2013.

[15] D Pongas, A Billard, and S Schaal. Rapid synchronization and accurate phase-locking of rhythmic motor primitives. *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 2911–2916, 2005.

[16] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008.

[17] Katharina Muelling, Jens Kober, and Jan Peters. Learning table tennis with a mixture of motor primitives. *IEEE-RAS International Conference on Humanoid Robots*, pages 411–416, 2010.

[18] J Peters, K Muelling, J Kober, D Nguyen-Tuong, and O Kroemer. Towards motor skill learning for robotics. In *International Symposium on Robotics Research (ISRR)*, 2010.

[19] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems (NIPS)*, volume 19, 2007.

[20] Andres El-Fakdi and Marc Carreras. Two-step gradient-based reinforcement learning for underwater robotics behavior learning. *Robotics and Autonomous Systems*, 61(3):271–282, 2013.

[21] Sham M Kakade, Michael J Kearns, and John Langford. Exploration in Metric State Spaces. *Proceedings of the 20th International Conference on Machine Learning (ICML)*, pages 306–312, 2003.

[22] Marc P Deisenroth and Carl Edward Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

[23] Marc P Deisenroth, Carl E Rasmussen, and Dieter Fox. Learning to control a low-cost manipulator using data-efficient reinforcement learning. In *Robotics: Science and Systems VII*, pages 57–64. 2011.

[24] Lilyana Mihalkova and Raymond Mooney. Using Active Relocation to Aid Reinforcement Learning. *Proceedings of the 19th International FLAIRS Conference*, (May):580–585, 2006.

[25] J. Buchli, F. Stulp, E. Theodorou, and S. Schaal. Learning variable impedance control. *The International Journal of Robotics Research*, 30(7):820–833, 2011.

[26] Andreas Vlachos. A stopping criterion for active learning. *Computer Speech & Language*, 22(3):295–312, 2008.

[27] Burr Settles. Active Learning Literature Survey. *Machine Learning*, 15:201–221, 2010.

[28] H S Seung, M Opper, and H Sompolinsky. Query by Committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 287–294, New York, NY, USA, 1992. ACM.

[29] Nicholas Roy and Andrew McCallum. Toward Optimal Active Learning through Sampling Estimation of Error Reduction. In *Proceedings of the 18th International Conference on Machine Learning*, ICML '01, pages 441–448, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[30] Yuhong Guo, Russ Greiner, and Canada Tg. Optimistic Active Learning using Mutual Information. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 823–829, 2007.

[31] Robert Moskovitch, Nir Nissim, Dima Stopel, Clint Feher, Roman Englert, and Yuval Elovici. Improving the detection of unknown computer worms activity using active learning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4667 LNAI:489–493, 2007.

[32] David A. Cohn. Neural Network Exploration Using Optimal Experiment Design. *Neural Networks*, 9(6):1071–1083, 1996.

[33] Burr Settles, Mark Craven, and Soumya Ray. Multiple-Instance Active Learning. In J C Platt, D Koller, Y Singer, and S T Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1289–1296. Curran Associates, Inc., 2008.

[34] Manuel Muhlig, Michael Gienger, Sven Hellbach, Jochen J. Steil, and Christian Goerick. Task-level imitation learning using variance-based movement optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1177–1184, 2009.

[35] David M. Rosen, Michael Kaess, and John J. Leonard. An incremental trust-region method for Robust online sparse least-squares estimation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1262–1269, 2012.

[36] Joshua C Bongard and Hod Lipson. Nonlinear System Identification Using Coevolution of Models and Tests. *Evolutionary Computation, IEEE Transactions on*, 9(4):361–384, aug 2005.

[37] Josh Bongard, Victor Zykov, and Hod Lipson. Resilient Machines Through Continuous Self-Modeling. *Science (New York, N.Y.)*, 314(5802):1118–1121, 2006.

[38] Krishnanand N Kaipa, Joshua C Bongard, and Andrew N Meltzoff. Self discovery enables robot social cognition: Are you my teacher? *Neural Networks*, 23(8â9):1113–1124, 2010.

[39] Matthias Rolf and Jochen J. Steil. Efficient exploratory learning of inverse kinematics on a bionic elephant trunk. *IEEE Transactions on Neural Networks and Learning Systems*, 25(6):1147–1160, 2014.

[40] S Otte, J Kulick, M Toussaint, and O Brock. Entropy-based strategies for physical exploration of the environment's degrees of freedom. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 615–622, sep 2014.

[41] O. B. Kroemer, R. Detry, J. Piater, and J. Peters. Combining active learning and reactive control for robot grasping. *Robotics and Autonomous Systems*, 58(9):1105–1116, 2010.

[42] Chris Lovell, Gareth Jones, Klaus-Peter Zauner, and Steve R. Gunn. Exploration and exploitation with insufficient resources. *JMLR: Workshop and Conference Proceedings*, 26:37–61, 2012.

[43] R Saegusa, G Metta, G Sandini, and S Sakka. Active motor babbling for sensorimotor learning. In *IEEE International Conference on Robotics and Biomimetics*, pages 794–799, feb 2008.

[44] Adrien Baranes and Pierre Yves Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73, 2013.

[45] Botond Bocsi, Lehel Csato, and Jan Peters. Alignment-based transfer learning for robot models. In *Proceedings of the International Joint Conference on Neural Networks*, 2013.

[46] Teodor Mihai Moldovan, Sergey Levine, Michael I Jordan, and Pieter Abbeel. Optimism-Driven Exploration for Nonlinear Systems. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3239–3246, 2015.

[47] Bruno Castro da Silva, George Konidaris, and Andrew G Barto. Learning Parameterized Skills. *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 1679–1686, 2012.

[48] Alan Broun, Chris Beck, Tony Pipe, Majid Mirmehdi, and Chris Melhuish. Bootstrapping a robot's kinematic model. *Robotics and Autonomous Systems*, 62(3):330–339, 2014.

[49] C Rosales, A Ajoudani, M Gabiccini, and A Bicchi. Active gathering of frictional properties from objects. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 3982–3987, sep 2014.

[50] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6):233–242, 1999.

[51] Aude Billard, Sylvain Calinon, R\ddotudiger Dillmann, and Stefan Schaal. Robot Programming by Demonstration. In Bruno Siciliano and Oussama Khatib, editors, *Springer Handbook of Robotics*, pages 1371–1394. Springer Berlin Heidelberg, 2008.

[52] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[53] Baris Akgun, Maya Cakmak, Karl Jiang, and Andrea L Thomaz. Keyframe-based Learning from Demonstration. *International Journal of Social Robotics*, 4(4):343–355, 2012.

[54] Kaijen Hsiao and Tomás Lozano-Pérez. Imitation learning of whole-body grasps. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5657–5662, 2006.

[55] Yun Lin and Yu Sun. Robot grasp planning based on demonstrated grasp strategies. *The International Journal of Robotics Research*, 34(1):26–42, 2015.

[56] Jacopo Aleotti and Stefano Caselli. Learning Manipulation Tasks from Human Demonstration and 3D Shape Segmentation. *Advanced Robotics*, 26(16):1863–1884, 2012.

[57] Sylvain Calinon, Florent D'halluin, Eric L. Sauser, Darwin G. Caldwell, and Aude G. Billard. A probabilistic approach based on dynamical systems to learn and reproduce gestures by imitation. *IEEE Robotics and Automation Magazine*, 17(January 2016):44–54, 2010.

[58] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.

[59] Gowrishankar Ganesh and Etienne Burdet. Motor planning explains human behaviour in tasks with multiple solutions. *Robotics and Autonomous Systems*, 61(4):362–368, 2013.

[60] S M Khansari-Zadeh and A Billard. Learning Stable Nonlinear Dynamical Systems With Gaussian Mixture Models. *Robotics, IEEE Transactions on*, 27(5):943–957, oct 2011.

[61] Mrinal Kalakrishnan, Peter Pastor, Ludovic Righetti, and Stefan Schaal. Learning objective functions for manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1331–1336, 2013.

[62] Andrej Gams, Bojan Nemec, Auke Jan Ijspeert, and Aleš Ude. Coupling movement primitives: Interaction with the environment and bimanual tasks. *IEEE Transactions on Robotics*, 30(4):816–830, 2014.

[63] Ales Ude, Andrej Gams, Tamim Asfour, and Jun Morimoto. Task-specific generalization of discrete and periodic dynamic movement primitives. *Robotics, IEEE Transactions on*, 26(5):800–815, 2010.

[64] J Aleotti and S Caselli. Robust trajectory learning and approximation for robot programming by demonstration. *Robotics and Autonomous Systems*, 54(5):409–413, 2006.

[65] Sylvain Calinon, Petar Kormushev, and Darwin G Caldwell. Compliant skills acquisition and multi-optima policy search with EM-based reinforcement learning. *Robotics and Autonomous Systems*, 61(4):369–379, 2013.

[66] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. A Generalized Path Integral Control Approach to Reinforcement Learning. *Journal of Machine Learning Research*, 11:3137–3181, dec 2010.

[67] Jens Kober and Jan Peters. Policy Search for Motor Primitives in Robotics. *Machine Learning*, 84(1-2):171–203, 2011.

[68] Pieter Abbeel, Adam Coates, and Andrew Y Ng. Autonomous Helicopter Aerobatics through Apprenticeship Learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[69] P Abbeel, Dmitri Dolgov, A Y Ng, and S Thrun. Apprenticeship learning for motion planning with application to parking lot navigation. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 1083–1090, sep 2008.

[70] Jason Chen and Alex Zelinsky. Programing by demonstration: coping with suboptimal teaching actions. *The International Journal of Robotics Research*, 22(5):299–319, 2003.

[71] Monica N Nicolescu and Maja J Mataric. Natural Methods for Robot Task Learning: Instructive Demonstrations, Generalization and Practice. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '03, pages 241–248, 2003.

[72] Ashesh Jain, Brian Wojcik, Thorsten Joachims, and Ashutosh Saxena. Learning Trajectory Preferences for Manipulators via Iterative Improvement. In C J C Burges, L Bottou, M Welling, Z Ghahramani, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 575–583. 2013.

[73] Daniel H Grollman and Aude G Billard. Robot learning from failed demonstrations. *International Journal of Social Robotics*, 4(4):331–342, 2012.

[74] Duy Nguyen-Tuong and Jan Peters. Model learning for robot control: a survey. *Cognitive Science*, 12(4):319–340, 2011.

[75] Axel Rottmann and Wolfram Burgard. Adaptive autonomous control using online value iteration with gaussian processes. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2106–2111, 2009.

[76] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Boston, Massachusetts, United States, 2006.

[77] Duy Nguyen-Tuong and Jan Peters. Local Gaussian process regression for real-time model-based robot control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 380–385, 2008.

[78] Franziska Meier, Philipp Hennig, and Stefan Schaal. Incremental Local Gaussian Regression. In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, number M, pages 972–980, 2014.

[79] Arjan Gijsberts and Giorgio Metta. Real-time model learning using Incremental Sparse Spectrum Gaussian Process Regression. *Neural Networks*, 41(0):59–69, 2013.

[80] Lehel Csato and Manfred Opper. Sparse Online Gaussian Processes. *Neural computation*, 2002.

[81] Daniel H Grollman and Odest Chadwicke Jenkins. Sparse incremental learning for interactive robot control policy estimation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3315–3320, 2008.

[82] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. Locally Weighted Learning. *Artificial Intelligence Review*, 11:11–73, 1997.

[83] M Arif, T Ishihara, and H Inooka. Incorporation of experience in iterative learning controllers using locally weighted learning. *Automatica*, 37(6):881–888, 2001.

[84] Christopher Lehnert and Gordon Wyeth. Locally Weighted Learning Model Predictive Control for Nonlinear and Time Varying Dynamics. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2619–2625, 2013.

[85] Lorenzo Jamone, Bruno Damas, and José Santos-Victor. Incremental learning of context-dependent dynamic internal models for robot control. In *Intelligent Control (ISIC), 2014 IEEE International Symposium on*, 2014.

[86] Tadej Petrič, Andrej Gams, Leon Žlajpah, and Aleš Ude. Online learning of task-specific dynamics for periodic tasks. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 1790–1795, 2014.

[87] Sethu Vijayakumar, Aaron D'Souza, and Stefan Schaal. Incremental online learning in high dimensions. *Neural computation*, 17:2602–2634, 2005.

[88] Jo-Anne Ting, Sethu Vijayakumar, and Stefan Schaal. Locally Weighted Regression for Control. In *Encyclopedia of Machine Learning*, volume 11, pages 613–624. Springer, 2010.

[89] Grzegorz Pajak and Iwona Pajak. Sub-optimal trajectory planning for mobile manipulators. *Robotica*, 33(06):1181–1200, 2015.

[90] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. Learning policy improvements with path integrals. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

[91] Igor Mordatch and Emanuel Todorov. Combining the benefits of function approximation and trajectory optimization. In *Robotics: Science and Systems (RSS)*, 2014.

[92] Fares J. Abu-Dakka, Francisco J. Valero, Jose Luis Suner, and Vicente Mata. A direct approach to solving trajectory planning problems using genetic algorithms with dynamics considerations in complex environments. *Robotica*, 33(3):669–683, 2015.

[93] Beomjoon Kim, Albert Kim, Hongkai Dai, Leslie Pack Kaelbling, and Tomas Lozano-perez. Generalizing over Uncertain Dynamics for Online Trajectory Generation. In *International Symposium on Robotics Research (ISRR)*, 2015.

[94] Chonhyon Park, Jia Pan, and Dinesh Manocha. High-DOF Robots in Dynamic Environments using Incremental Trajectory Optimization. *International Journal of Humanoid Robotics*, 11(02), 2014.

[95] Michael Posa and Russ Tedrake. Direct Trajectory Optimization of Rigid Body Dynamical Systems Through Contact. In *Algorithmic Foundations of Robotics X*, volume 86, pages 527–542, 2013.

[96] Jingru Luo and Kris Hauser. Robust Trajectory Optimization Under Frictional Contact with Iterative Learning. In *Robotics: Science and Systems (RSS)*, 2015.

[97] Yajia Zhang, Jingru Luo, and Kris Hauser. Sampling-based motion planning with dynamic intermediate state objectives: Application to throwing. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2551–2556, 2012.

[98] Eric Aboaf, Christopher G. Atkeson, and David J. Reinkensmeyer. Task Level Robot Learning: Ball Throwing. Technical report, MIT, Cambridge, MA, 1987.

[99] Michael S. Branicky, Ross A. Knepper, and James J. Kuffner. Path and trajectory diversity: Theory and algorithms. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1359–1364, 2008.

[100] Dmitry Berenson, Pieter Abbeel, and Ken Goldberg. A robot path planning framework that learns from experience. In *International Conference on Robotics and Automation (ICRA)*, pages 3671–3678, 2012.

[101] Chris Bowen, Gu Ye, and Ron Alterovitz. Asymptotically optimal motion planning for learned tasks using time-dependent cost maps. *IEEE Transactions on Automation Science and Engineering*, 12(1):171–182, 2015.

[102] Peter Pastor, H Hoffmann, T Asfour, and S Schaal. Learning and generalization of motor skills by learning from demonstration. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 763–768, may 2009.

[103] Minija Tamosiunaite, Bojan Nemec, Aleš Ude, and Florentin Wörgötter. Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives. *Robotics and Autonomous Systems*, 59(11):910–922, 2011.

[104] Bojan Nemec, Denis Forte, Rok Vuga, Minija Tamosiunaite, Florentin Worgotter, and Ales Ude. Applying statistical generalization to determine search direction for reinforcement learning of movement primitives. In *IEEE-RAS International Conference on Humanoid Robots*, pages 65–70, 2012.

[105] Leonel Rozo, Pablo Jimenez, and Carme Torras. Force-based robot learning of pouring skills using parametric hidden Markov models. *International Workshop on Robot Motion and Control, RoMoCo*, pages 227–232, jul 2013.

[106] Sascha Brandi, Oliver Kroemer, and Jan Peters. Generalizing Pouring Actions Between Objects using Warped Parameters. In *Humanoid Robots, 2014 14th IEEE-RAS International Conference on*, 2014.

[107] Oliver Kroemer, E Ugur, E Oztop, and Jan Peters. A Kernel-based Approach to Direct Action Perception. *International Conference on Robotics and Automation*, pages 2605–2610, 2012.

[108] Akihiko Yamaguchi, Christopher G. Atkeson, and Tsukasa Ogasawara. Pouring Skills with Planning and Learning Modeled from Human Demonstrations. *International Journal of Humanoid Robotics*, 12(03):1550030, 2015.

[109] Akihiko Yamaguchi, Christopher G Atkeson, Scott Niekum, and Tsukasa Ogasawara. Learning Pouring Skills from Demonstration and Practice. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 908–915, 2014.

[110] Gisbert Lawitzky. A Navigation System for Cleaning Robots. *Autonomous Robots*, 9(3):255–260, 2000.

[111] J.L. Jones. Robots at the tipping point: the road to iRobot Roomba. *IEEE Robotics and Automation Magazine*, 13(1):1–3, 2006.

[112] Chang Doo Jung, Won Jee Chung, Jin Su Ahn, Myung Sik Kim, Gi Soo Shin, and Soon Jea Kwon. Optimal mechanism design of in-pipe cleaning robot. In *IEEE International Conference on Mechatronics and Automation*, pages 1327–1332, 2011.

[113] Jurgen Hess, Gian Diego Tipaldi, and Wolfram Burgard. Null space optimization for effective coverage of 3D surfaces using redundant manipulators. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1923–1928, 2012.

[114] Jurgen Hess, Maximilian Beinhofer, and Wolfram Burgard. A probabilistic approach to high-confidence cleaning guarantees for low-cost cleaning robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5600–5605, 2014.

[115] Martin Do, Julian Schill, Johannes Ernesti, and Tamim Asfour. Learn to Wipe : A Case Study of Structural Bootstrapping from Sensorimotor Experience. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1858–1864, 2014.

[116] Chih-Hung King, Tiffany L. Chen, Advait Jain, and Charles C. Kemp. Towards an assistive robot that autonomously performs bed baths for patient hygiene. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 319–324, 2010.

[117] Ariyan M. Kabir, Joshua D. Langsfeld, Cunbo Zhuang, Krishnanand N. Kaipa, and Satyandra K. Gupta. Automated Learning of Operation Parameters for Robotic Cleaning by Mechanical Scrubbing. In *ASME Manufacturing Science and Engineering Conference2*, 2016.

[118] Ariyan M. Kabir, Joshua D. Langsfeld, Shaurya Shriyam, Vinaichandra Sai Rachakonda, Cunbo Zhuang, Krishnanand N. Kaipa, Jeremy Marvel, and Satyandra K. Gupta. Planning algorithms for multi-setup multi-pass robotic cleaning with oscillatory moving tools. In *IEEE International Conference on Automation Science and Engineering (CASE)*, pages 751–757, 2016.

[119] Fouad F. Khalil and Pierre Payeur. Dexterous Robotic Manipulation of Deformable Objects with Multi-Sensory Feedback - a Review. *Robot Manipulators, Trends and Development*, pages 587–621, 2010.

[120] P Jiménez. Survey on model-based manipulation planning of deformable objects. *Robotics and Computer-Integrated Manufacturing*, 28(2):154–163, 2012.

[121] Matthew Bell. *Flexible Object Manipulation*. PhD thesis, Dartmouth College, 2010.

[122] Mitul Saha and Pekka Isto. Manipulation planning for deformable linear objects. *IEEE Transactions on Robotics*, 23(6):1141–1150, 2007.

[123] Mark Moll and Lydia E Kavraki. Path Planning for Deformable Linear Objects. *IEEE Transactions on Robotics*, 22(4):625–636, 2006.

[124] Ankit J. Shah and Julie A. Shah. Towards Manipulation Planning for Multiple Interlinked Deformable Linear Objects. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3908–3915, 2016.

[125] Martin Stommel and Weiliang Xu. Learnability of the Moving Surface Profiles of a Soft Robotic Sorting Table. *IEEE Transactions on Automation Science and Engineering*, 13(4):1581–1587, 2016.

[126] Madusudanan Sathia Narayanan, Xiaobo Zhou, Sudha Garimella, Wayne Waz, Frank Mendel, and Venkat N. Krovi. Data driven development of haptic models for needle biopsy phantoms. In *ASME 2012 5th Annual Dynamic Systems and Control Conference joint with the JSME 2012 11th Motion and Vibration Conference*, 2012.

[127] Stéphane Cotin, Hervé Delingette, and Nicholas Ayache. A hybrid elastic model for real-time cutting, deformations, and force feedback for surgery training and simulation. *The Visual Computer*, 16(8):437–452, 2000.

[128] Celine Paloc, Fernando Bello, Richard I. Kitney, and Ara Darzi. Online multiresolution volumetric mass spring model for real time soft tissue deformation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2002*, pages 219–226. 2002.

[129] T. M. Caldwell, D. Coleman, and N. Correll. Optimal Parameter Identification for Discrete Mechanical Systems with Application to Flexible Object Manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 898–905, 2014.

[130] W. Mollemans, F. Schutyser, N. Nadjmi, F. Maes, and P. Suetens. Predicting soft tissue deformations for a maxillofacial surgery planning system: From computational strategies to a complete clinical validation. *Medical Image Analysis*, 11(3):282–301, 2007.

[131] Tirupathi R. Chandrupatla, Ashok D. Belegundu, T. Ramesh, and Chaitali Ray. *Introduction to Finite Elements in Engineering*. Prentice Hall, Upper Saddle River, 1997.

[132] Wouter Mollemans, Filip Schutyser, Nasser Nadjmi, and Paul Suetens. Very fast soft tissue predictions with mass tensor model for maxillofacial surgery planning systems. *International Congress Series*, 1281(CARS 2005: Computer Assisted Radiology and Surgery):491–496, 2005.

[133] Zachary Pezzementi, Daniel Ursu, Sarthak Misra, and Allison M. Okamura. Modeling Realistic Tool-Tissue Interactions with Haptic Feedback: A Learning-based Method. In *Symposium on Haptic Interfaces for Virtual Environments and Teleoperator Systems*, pages 209–215, 2008.

[134] Desai Chen, David I. W. Levin, Shinjiro Sueda, and Wojciech Matusik. Data-Driven Finite Elements for Geometry and Material Design. *ACM Transactions on Graphics*, 34(4), 2015.

[135] Iasonas Kokkinos and Alan Yuille. Unsupervised learning of object deformation models. In *Proceedings of the IEEE International Conference on Computer Vision*, 2007.

[136] J. L. Pedreño-Molina, A. Guerrero-González, J. Calabozo-Moran, J. López-Coronado, and P. Gorce. A neural tactile architecture applied to real-time stiffness estimation for a large scale of robotic grasping systems. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 2007.

[137] Barbara Frank, Rudiger Schmedding, Cyrill Stachniss, Matthias Teschner, and Wolfram Burgard. Learning the elasticity parameters of deformable objects with a manipulation robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1877–1883, 2010.

[138] Fernanda Coutinho and Rui Cortesao. Comparison of position and force-based techniques for environment stiffness estimation in robotic tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4933–4938, 2012.

[139] Ana Maria Cretu, Pierre Payeur, and Emil M. Petriu. Soft object deformation monitoring and learning for model-based robotic hand manipulation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 42(3):740–753, 2012.

[140] R. E. Goldman, A. Bajo, and N. Simaan. Algorithms for autonomous exploration and estimation in compliant environments. *Robotica*, 31:71–87, 2013.

[141] Elif Ayvali, Rangaprasad Arun Srivatsan, Long Wang, Rajarshi Roy, Nabil Simaan, Howie Choset, and Et Al. Using Bayesian Optimization to Guide Probing of a Flexible Environment for Simultaneous Registration and Stiffness Mapping. *ACM Transactions on Graphics*, 2(4):1–27, 2015.

[142] David Navarro-Alarcon, Hiu Man Yip, Zerui Wang, Yun-Hui Liu, Fangxun Zhong, Tianxue Zhang, and Peng Li. Automatic 3-D Manipulation of Soft Objects by Robotic Arms With an Adaptive Deformation Model. *IEEE Transactions on Robotics*, 32(2):429–441, 2016.

[143] Dmitry Berenson. Manipulation of deformable objects without modeling and simulating deformation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4525–4532, 2013.

[144] Henry Lu. *Learning force-based manipulation of deformable objects from multiple demonstrations*. PhD thesis, University of California at Berkeley, 2015.

[145] Alex X. Lee, Sandy H. Huang, Dylan Hadfield-Menell, Eric Tzeng, and Pieter Abbeel. Unifying scene registration and trajectory optimization for learning from demonstrations with application to manipulation of deformable objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4402–4407, 2014.

[146] Alex X. Lee, Abhishek Gupta, Henry Lu, Sergey Levine, and Pieter Abbeel. Learning from multiple demonstrations using trajectory-aware non-rigid registration with applications to deformable object manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5265–5272, 2015.

[147] Alexander M. Schmidts, Dongheui Lee, and Angelika Peer. Imitation learning of human grasping skills from motion and force data. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1002–1007, 2011.

[148] Mohammad Khansari, Ellen Klingbeil, and Oussama Khatib. Adaptive human-inspired compliant contact primitives to perform surface-surface contact under uncertainty. *The International Journal of Robotics Research*, 35(13):1651–1675, 2016.

[149] Leonel Rozo. *Robot Learning from Demonstration of Force-based Manipulation Tasks*. Doctoral, Polytechnic University of Catalonia, 2013.

[150] Tesca Fitzgerald, Kalesha Bullard, Andrea Thomaz, and Ashok Goel. Situated Mapping for Transfer Learning. *Advances in Cognitive Systems*, 4, 2016.

[151] Craig E Birkhimer. *Extracting human strategies for use in robotic assembly*. PhD thesis, Case Western Reserve University, 2005.

[152] Marcela Poffald, Yajia Zhang, and Kris Hauser. Learning problem space metrics for motion primitive selection. In *Machine Learning in Planning and Control of Robot Motion Workshop: IROS 2014*, 2014.

[153] Vasiliki Koropouli, Sandra Hirche, and Dongheui Lee. Generalization of Force Control Policies from Demonstrations for Constrained Robotic Motion Tasks. *Journal of Intelligent Robot Systems*, 2015.

[154] Joshua D. Langsfeld, Krishnanand N. Kaipa, Rodolphe J. Gentili, James A. Reggia, and Satyandra K. Gupta. Incorporating Failure-to-Success Transitions in Imitation Learning for a Dynamic Pouring Task. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS): Workshop on Compliant Manipulation*, 2014.

[155] C G Atkeson and S Schaal. Learning tasks from a single demonstration. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 2, pages 1706–1712 vol.2, apr 1997.

[156] Chrystopher L Nehaniv and Kerstin Dautenhahn. The Correspondence Problem. In *Imitation in animals and artifacts*, chapter 2, page 41. The MIT Press, 2002.

[157] Marcel Brass and Cecilia Heyes. Imitation: is cognitive neuroscience solving the correspondence problem? *Trends in Cognitive Sciences*, 9(10):489–495, 2005.

[158] P. M. Fitts and M. I. Posner. *Human Performance*. Brooks/Cole, Oxford, England, 1967.

[159] Julien Doyon and Habib Benali. Reorganization and plasticity in the adult brain during learning of motor skills. *Current Opinion in Neurobiology*, 15:161–167, 2005.

[160] Saroj Saimek and Perry Y Li. Motion Planning and Control of a Swimming Machine. *The International Journal of Robotics Research*, 23(1):27–53, 2004.

[161] S Garrido-Jurado, R Muoz-Salinas, F J Madrid-Cuevas, and M J Marn-Jimnez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 2014.

[162] E Guizzo and E Ackerman. The rise of the robot worker. *Spectrum, IEEE*, 49(10):34–41, oct 2012.

[163] Joshua D. Langsfeld, Krishnanand N. Kaipa, and Satyandra K. Gupta. Generation and Exploitation of Local Models for Rapid Learning of a Pouring Task. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS): Second Machine Learning in Planning and Control of Robot Motion Workshop*, 2015.

[164] Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias Seeger. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *Proceedings of the 27th International Conference on Machine Learning (ICML 2010)*, pages 1015–1022, 2010.

[165] John Schulman, Sergey Levine, Michael Jordan, and Pieter Abbeel. Trust Region Policy Optimization. In *International Conference on Machine Learning (ICML)*, 2015.

[166] Mark F. Bear, Barry W. Connors, and Michael A. Paradiso. *Neuroscience: Exploring the Brain*. Lippincott Williams and Wilkins, 3rd edition, 2006.

[167] Steven G. Johnson. The NLopt nonlinear-optimization package.

[168] Joshua D. Langsfeld, Ariyan M. Kabir, Krishnanand N. Kaipa, and Satyandra K. Gupta. Online Learning of Part Deformation Models for Robotic Cleaning. In *ASME Manufacturing Science and Engineering Conference*, 2016.

[169] J. Kober, J. a. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32:1238–1274, 2013.

[170] Joshua D. Langsfeld, Ariyan M. Kabir, Krishnanand N. Kaipa, and Satyandra K. Gupta. Robotic Bimanual Cleaning of Deformable Objects with Online Learning of Part and Tool Models. In *IEEE International Conference on Automation Science and Engineering (CASE)*, 2016.

[171] Krishnanand N. Kaipa, Srudeep S. Thevendria-Karthic, Shaurya Shriyam, Ariyan M. Kabir, Joshua D. Langsfeld, and Satyandra K. Gupta. Resolving automated perception system failures in bin-picking tasks using assistance from remote human operators. In *IEEE International Conference on Automation Science and Engineering (CASE)*, 2015.

[172] Carlos Morato, Krishnanand N. Kaipa, and Satyandra K. Gupta. Improving assembly precedence constraint generation by utilizing motion planning and part interaction clusters. *Computer-Aided Design*, 45(11), 2013.